

Intel·ligència artificial avançada

Raúl Benítez

Gerard Escudero

Samir Kanaan

PID_00174125

Els textos i imatges publicats en aquesta obra estan subjectes –llevat que s'indiqui el contrari– a una llicència de Reconeixement-NoComercial-SenseObraDerivada (BY-NC-ND) v.3.0 Espanya de Creative Commons. Podeu copiar-los, distribuir-los i transmetre'ls públicament sempre que en citeu l'autor i la font (FUOC. Fundació per a la Universitat Oberta de Catalunya), no en feu un ús comercial i no en feu obra derivada. La llicència completa es pot consultar a <http://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.ca>.

Índex

Introducció	7
1. Introducció a la intel·ligència artificial (IA)	9
1.1. Neurones i transistors	9
1.2. Breu història de la IA	12
1.3. Àmbits d'aplicació de la intel·ligència artificial.....	15
2. Recomanadors i agrupaments	19
2.1. Mètriques i mesures de similitud.....	20
2.1.1. Exemple d'aplicació	20
2.1.2. Distància euclidiana	21
2.1.3. Correlació de Pearson	23
2.1.4. Processament de dades reals	25
2.1.5. Conclusions	25
2.2. Recomanadors basats en memòria	26
2.2.1. Conceptes generals	26
2.2.2. Aproximacions simples	26
2.2.3. Recomanació ponderada	27
2.2.4. Conclusions	28
2.3. Algorismes d'agrupament (<i>clustering</i>)	29
2.3.1. Exemple d'aplicació	29
2.3.2. Conceptes generals	31
2.3.3. Agrupament jeràrquic. Dendrogrames	32
2.3.4. <i>k</i> -mitjanes (<i>k-means</i>)	35
2.3.5. <i>c</i> -mitjanes difús (<i>fuzzy c-means</i>).....	37
2.3.6. Agrupament espectral (<i>spectral clustering</i>)	37
2.3.7. Recomanadors basats en models.....	39
3. Extracció i selecció d'atributs	40
3.1. Tècniques de factorització matricial.....	42
3.1.1. Descomposició en valors singulars (SVD)	43
3.1.2. Anàlisi de components principals (PCA).....	47
3.1.3. Anàlisi de components independents (ICA)	61
3.1.4. Factorització de matrius no negatives (NMF)	72
3.2. Discriminació de dades en classes	79
3.2.1. Anàlisi de discriminants lineals (LDA)	79
3.3. Visualització de dades multidimensionals	86
3.3.1. Escalament multidimensional (MDS)	86
4. Classificació	93
4.1. Introducció	93

4.1.1.	Categorització de textos	94
4.1.2.	Aprendentatge automàtic per a classificació	96
4.1.3.	Tipologia d'algorismes per a classificació	97
4.2.	Mètodes basats en models probabilístics	98
4.2.1.	Naïve Bayes.....	98
4.2.2.	Màxima entropia	102
4.3.	Mètodes basats en distàncies	104
4.3.1.	kNN	105
4.3.2.	Classificador lineal basat en distàncies	108
4.3.3.	<i>Clustering</i> dins de classes.....	110
4.4.	Mètodes basats en regles	112
4.4.1.	Arbres de decisió.....	112
4.4.2.	AdaBoost	120
4.5.	Classificadors lineals i mètodes basats en <i>kernels</i>	125
4.5.1.	Classificador lineal basat en producte escalar	125
4.5.2.	Classificador lineal amb <i>kernel</i>	130
4.5.3.	<i>Kernels</i> per a tractament de textos	135
4.5.4.	Màquines de vectors de suport	143
4.6.	Protocols de test	157
4.6.1.	Protocols de validació	157
4.6.2.	Mesures d'avaluació	158
4.6.3.	Tests estadístics	160
4.6.4.	Comparativa de classificadors	161
5.	Optimització	164
5.1.	Introducció	164
5.1.1.	Tipologia dels mètodes d'optimització.....	167
5.1.2.	Característiques dels metaheurístics d'optimització ..	167
5.2.	Optimització mitjançant multiplicadors de Lagrange	168
5.2.1.	Descripció del mètode	169
5.2.2.	Exemple d'aplicació	170
5.2.3.	Anàlisi del mètode.....	171
5.3.	Recocció simulada	171
5.3.1.	Descripció del mètode	172
5.3.2.	Exemple d'aplicació	173
5.3.3.	Anàlisi del mètode.....	175
5.3.4.	Codi font en Python	176
5.4.	Algorismes genètics	178
5.4.1.	Descripció del mètode	179
5.4.2.	Ampliacions i millores.....	181
5.4.3.	Exemple d'aplicació	181
5.4.4.	Anàlisi del mètode.....	182
5.4.5.	Codi font en Python	183
5.5.	Colònies de formigues	185
5.5.1.	Descripció del mètode	186
5.5.2.	Exemple d'aplicació	187
5.5.3.	Anàlisi del mètode.....	190

5.5.4. Codi font en Python	191
5.6. Optimització amb eixams de partícules	193
5.6.1. Descripció del mètode	194
5.6.2. Exemple d'aplicació	197
5.6.3. Anàlisi del mètode.....	197
5.6.4. Codi font en Python	199
5.7. Cerca tabú	201
5.7.1. Descripció del mètode	201
5.7.2. Exemple d'aplicació	202
5.7.3. Anàlisi del mètode.....	204
5.7.4. Codi font en Python	205
6. Annex: conceptes bàsics d'estadística	208
Activitats	211
Bibliografia	213

Introducció

Aquest mòdul està organitzat de la manera següent: els mètodes de cerca i optimització es descriuen en l'apartat 2, on es detallaran les tècniques d'extracció d'informació de bases de dades que continguin informació semàntica, com per exemple webs de notícies o les converses entre diversos membres d'una xarxa social.

Les tècniques de caracterització de dades s'estudiaran en l'apartat 3, i s'hi descriuran les tècniques principals basades en descomposició de les dades en modes principals. En l'apartat 3 també s'estudiaran les tècniques d'extracció de característiques i un mètode de visualització de dades multidimensionals.

Els algorismes de classificació de dades es presenten en l'apartat 4, en el qual s'estudiaran els principals mètodes de classificació i reconeixement de patrons.

En l'apartat 5 s'expliquen algunes tècniques avançades d'intel·ligència evolutiva, algorismes que utilitzen regles heurístiques inspirades en el funcionament evolutiu dels sistemes biològics.

1. Introducció a la intel·ligència artificial (IA)

1.1. Neurones i transistors

Començarem plantejant la pregunta filosòfica fonamental, i així podrem dedicar els nostres esforços a aspectes de caràcter científicotècnic.

És físicament possible que una màquina presenti capacitat d'abstracció similar a la intel·ligència humana?

Per a respondre aquesta pregunta, cal tenir en compte que el cervell humà és el sistema de reconeixement de patrons més complex i eficient que coneixem. Els humans fem accions tan sorprenents com identificar un conegut entre la multitud o reconèixer d'oïda el solista d'un concert per a violí. En el cervell humà, les funcions cognitives es fan mitjançant l'activació coordinada d'uns 90.000.000.000 de cèl·lules nervioses interconnectades mitjançant enllaços sinàptics. L'activació neuronal segueix complexos processos biofísics que garanteixen un funcionament robust i adaptatiu, i ens permet fer funcions com el processament d'informació sensorial, la regulació fisiològica dels òrgans, el llenguatge o l'abstracció matemàtica.

La neurociència actual encara no aporta una descripció detallada sobre com l'activació individual de les neurones dóna lloc a la formació de representacions simbòliques abstractes. El que sí que sembla clar és que en la majoria de processos cognitius hi ha una separació d'escales entre la dinàmica a escala neuronal i l'aparició d'activitat mental abstracta. Aquesta separació d'escales representa la ruptura del vincle existent entre el maquinari (neurones) i el programari del nostre cervell (operacions abstractes, estats mentals), i constitueix la hipòtesi de partida perquè els símbols abstractes puguin ser manipulats per sistemes artificials que no requereixin un substrat fisiològic natural. La possibilitat de manipular expressions lògiques i esquemes abstractes mitjançant sistemes artificials és la que permet l'existència del que coneixem com a *intel·ligència artificial*.

Per descomptat, el cervell no és l'únic sistema físic en el qual es produeix una separació de la dinàmica a diferents escales. Aquesta característica també s'observa en molts altres sistemes complexos que presenten fenòmens d'autorganització no lineal. De la mateixa manera que és possible descriure els

Lectures complementàries

C. Koch (1999). *Biophysics of Computation: Information Processing in Single Neurons*. EUA: Oxford University Press.

corrents oceànics sense necessitat de referir-se al moviment microscòpic de les molècules d'aigua, el pensament abstracte es pot analitzar sense necessitat de referir-se l'activació elèctrica cerebral a escala neuronal.

En qualsevol cas, una de les qüestions de més rellevància i encara no resoltres de la neurociència actual és saber si hi ha processos mentals –com la consciència, l'empatia o la creativitat–, que estiguin intrínsecament lligats a la realitat biofísica del sistema nerviós humà i siguin, per tant, inaccessibles a un sistema artificial.

Un altre aspecte important en el funcionament del cervell humà és el paper que té l'experiència i l'aprenentatge. El cervell humà actual no és només resultat d'una evolució biològica basada en alteracions genètiques, sinó també del conjunt de tècniques i coneixements que la humanitat ha anat acumulant amb el temps. Aspectes com la cultura o el llenguatge, transmesos de generació en generació, també determinen la manera en la qual s'estableixen patrons d'activació neuronal en el nostre cervell, i per tant, contribueixen a l'emergència de processos d'abstracció en els quals es basen àrees com les matemàtiques o la literatura. A escala biològica, hi ha diversos mecanismes que permeten l'existència de processos d'activació neuronal dependents de l'experiència prèvia i de l'entrenament. El mecanisme principal és conegut com a *plasticitat sinàptica*, un fenomen pel qual les connexions sinàptiques entre neurones modulen la seva intensitat en funció de l'activitat que hagin experimentat prèviament. D'aquesta manera, com més vegades s'activi un cert canal d'activació neuronal, més fàcil resultarà activar-lo en el futur i integrar-lo a nous processos cognitius de més complexitat.

La plasticitat neuronal és la base de la majoria de processos d'aprenentatge i memòria. Aquest paradigma es coneix com a *aprenentatge reforçat*, ja que l'activitat sinàptica es reforça en funció del nombre de vegades que s'estableix una connexió entre neurones. Aquesta regla –que relaciona l'activitat neuronal amb la funció cognitiva–, es coneix com a *regla de Hebb* pels treballs del neuropsicòleg canadenc Donald O. Hebb publicats en el seu llibre de 1949 *The organization of behavior*. Algunes tècniques d'intel·ligència artificial com els mètodes d'*aprenentatge supervisat* també es basen en regles similars que permeten modificar de manera adaptativa la manera en què el sistema artificial processa la informació.

La intel·ligència artificial (IA) és una disciplina acadèmica relacionada amb la teoria de la computació que té l'objectiu d'emular algunes de les facultats intel·lectuals humans en sistemes artificials. Amb *intel·ligència humana* ens referim típicament a processos de percepció sensorial (visió, audició, etc.) i als processos consegüents de reconeixement de patrons, per la qual cosa les

aplicacions més habituals de la IA són el tractament de dades i la identificació de sistemes. Això no exclou que la IA, des dels seus inicis en la dècada del 1960, hi hagi resolt problemes de caràcter més abstracte com la demostració de teoremes matemàtics, l'adquisició del llenguatge, jugar a escacs o la traducció automàtica de textos. El disseny d'un sistema d'intel·ligència artificial normalment requereix la utilització d'eines de disciplines molt diferents com el càlcul numèric, l'estadística, la informàtica, el processament de senyals, el control automàtic, la robòtica o la neurociència. Per aquest motiu, malgrat que la intel·ligència artificial es considera una branca de la informàtica teòrica, és una disciplina a la qual contribueixen de manera activa nombrosos científics, tècnics i matemàtics. En alguns aspectes, a més, es beneficia d'investigacions en àrees tan diverses com la psicologia, la sociologia o la filosofia.

Malgrat que s'han produït nombrosos avanços en el camp de la neurociència des del descobriment de la neurona per part de Santiago Ramón y Cajal a la fi del segle XIX, les tecnologies actuals estan molt lluny de poder dissenyar i fabricar sistemes artificials de la complexitat del cervell humà. De fet, avui dia estem lluny de reproduir de manera sintètica les propietats electroquímiques de la membrana cel·lular d'una sola neurona. Però com hem comentat anteriorment, la manipulació de conceptes i expressions abstractes no està suportada a l'existència d'un sistema biològic de computació. En definitiva, un ordinador no és més que una màquina que processa representacions abstractes seguint unes regles predefinides.

Avanços de la microelectrònica

Actualment, l'única tecnologia que permet implementar sistemes d'intel·ligència artificial són els sistemes electrònics basats en dispositius d'estat sòlid com el transistor. En efecte, gràcies als grans avanços de la microelectrònica des dels anys setanta, els ordinadors actuals disposen d'una gran capacitat de càlcul que permet la implementació de sistemes avançats de tractament de dades i reconeixement de patrons. Els sistemes digitals basats en transistor constitueixen una tecnologia ràpida, robusta i de mida reduïda que permet l'execució seqüencial d'operacions aritmèticològiques. En moltes aplicacions concretes, els sistemes artificials poden arribar a presentar un rendiment notablement millor que el del cervell humà mateix. Aquest és el cas, per exemple, en aquelles situacions que requereixin gestionar grans quantitats de dades o que exigeixin una execució ràpida de càlculs matemàtics.

Un sistema d'intel·ligència artificial requereix una seqüència finita d'instruccions que especifiqui les diferents accions que executa la computadora per a resoldre un problema determinat. Aquesta seqüència d'instruccions constitueix l'*estructura algorítmica* del sistema d'intel·ligència artificial.

Es coneix com a *mètode efectiu o algorisme* al procediment per a trobar la solució a un problema mitjançant la reducció d'aquest a un conjunt de regles.

De vegades, els sistemes d'IA resolen problemes de manera *heurística* mitjançant un procediment d'assaig i error que incorpora informació rellevant basada en coneixements previs. Quan un mateix problema es pot resoldre mitjançant sistemes naturals (cervell) o artificials (computadora), els algorismes que segueix cada implementació solen ser completament diferents, ja que el conjunt d'instruccions elementals de cada sistema són també diferents. El cervell processa la informació mitjançant l'activació coordinada de xarxes de neurones en àrees especialitzades (còrtex visual, còrtex motor, etc.). En el sistema nerviós, les dades es transmeten i reben codificades en variables com la freqüència d'activació de les neurones o els intervals en els quals es generen els potencials d'acció neuronals. L'elevat nombre de neurones que intervenen en un procés de computació natural fa que les fluctuacions fisiològiques tinguin un paper rellevant i que els processos computacionals es facin de manera estadística mitjançant l'activitat amitjanada en subconjunts de neurones.

En un sistema IA, en canvi, les instruccions bàsiques són les pròpies d'una computadora, és a dir operacions aritmèticològiques, de lectura/escriptura de registres i de control de flux seqüencial. La taula 1 descriu les diferències fonamentals entre sistemes d'intel·ligència artificial i natural en les escales més rellevants.

Taula 1. Comparació entre intel·ligència natural i artificial a diferents nivells

Nivell	Natural	Artificial
Abstracció	Representació i manipulació d'objectes abstractes	Representació i manipulació d'objectes abstractes
Computacional	Activació coordinada d'àrees cerebrals	Algorisme / procediment efectiu
Programació	Connexions sinàptiques plasticitat	Seqüència d'operacions aritmèticològiques
Arquitectura	Xarxes excitatòries i inhibitòries	CPU + memòria
Maquinari	Neurona	Transistor

La idea principal és que, malgrat les enormes diferències entre sistemes naturals i artificials, a un cert nivell d'abstracció tots dos es poden descriure com a sistemes de processament d'objectes abstractes mitjançant un conjunt de regles.

1.2. Breu història de la IA

El naixement de la IA com a disciplina d'investigació es remunta al 1956, durant una conferència sobre informàtica teòrica que va tenir lloc al Dartmouth College (Estats Units). A aquesta conferència van assistir alguns dels científics

que posteriorment es van encarregar de desenvolupar la disciplina en diferents àmbits i de dotar-la d'una estructura teòrica i computacional apropiada. Entre els assistents hi havia John McCarthy, Marvin Minsky, Allen Newell i Herbert Simon. En la conferència, A. Newell i H. Simon van presentar un treball sobre demostració automàtica de teoremes que van denominar *Logic Theorist*. El Logic Theorist va ser el primer programa d'ordinador que emulava característiques pròpies del cervell humà, per la qual cosa és considerat el primer sistema d'intel·ligència artificial de la història. El sistema era capaç de demostrar gran part dels teoremes sobre lògica matemàtica que es presentaven en els tres volums dels *Principia Mathematica* d'Alfred N. Whitehead i Bertrand Russell (1910-1913).

Minsky i McCarthy van fundar més tard el laboratori d'intel·ligència artificial del Massachusetts Institute of Technology (MIT), un dels grups pioners en l'àmbit. L'activitat dels anys cinquanta és conseqüència de treballs teòrics d'investigadors anteriors com Charles Babbage (autor de la màquina analítica, 1842), Kurt Gödel (teorema d'incompletitud, 1930), Alan Turing (màquina universal, 1936), Norbert Wiener (cibernètica, 1943) i John von Neumann (arquitectura del computador, 1950). L'arquitectura de Newmann consta d'una unitat central de procés (CPU) i d'un sistema d'emmagatzematge de dades (memòria), i va ser utilitzada el 1954 per la RAND Corporation per a construir JOHNIAC (John v. Neumann Numerical Integrator and Automatic Computer), una de les primeres computadores en les quals més tard es van implementar sistemes d'intel·ligència artificial com el Logic Theorist de Newell i Simon.

El 1954 també va aparèixer l'IBM 704, la primera computadora de producció en cadena, i amb aquesta es van desenvolupar nombrosos llenguatges de programació específicament dissenyats per a implementar sistemes d'intel·ligència artificial com el LISP. Juntament amb aquests avanços, es van produir els primers intents per a determinar la presència de comportament intel·ligent en una màquina. El més rellevant des del punt de vista històric va ser proposat per Alan Turing en un article de 1950 publicat en la revista *Mind* i titulat "Computing Machinery and Intelligence".

En aquest treball es proposa un test d'intel·ligència per a màquines segons el qual una màquina presentaria un comportament intel·ligent en la mesura en què fos capaç de mantenir una conversa amb un humà sense que una altra persona pugui distingir qui és l'humà i qui l'ordinador. Encara que el *test de Turing* ha sofert innombrables adaptacions, correccions i controvèrsies, posa de manifest els primers intents d'assolir una definició objectiva de la intel·ligència.

En aquest context, és d'especial rellevància el *teorema d'incompletitud de Gödel* de 1931, un conjunt de teoremes de lògica matemàtica que estableixen les

limitacions inherents a un sistema basat en regles i procediments lògics (com són tots els sistemes d'IA).

Després dels primers treballs en IA dels anys cinquanta, en la dècada dels seixanta es va produir un gran esforç de formalització matemàtica dels mètodes utilitzats pels sistemes d'IA.

Els anys setanta, en part com a resposta al test de Turing, es va produir el naixement d'una àrea coneguda com a *processament del llenguatge natural* (NLP, *natural language processing*), una disciplina dedicada a sistemes artificials capaços de generar frases intel·ligents i de mantenir converses amb humans. L'NLP ha donat lloc a diverses àrees d'investigació en el camp de la lingüística computacional, incloent-hi aspectes com la desambiguació semàntica o la comunicació amb dades incompltes o errònies. Malgrat els grans avanços en aquest àmbit, continua sense haver-hi una màquina que pugui passar el test de Turing tal com es va plantejar en l'article original. Això no és tant a causa d'un fracàs de la IA com al fet que els interessos de l'àrea s'han anat redefinint al llarg de la història. El 1990, el controvertit empresari Hugh Loebner i el Cambridge Center for Behavioral Studies van instaurar el *premi Loebner*, un concurs anual certament heterodox en el qual es premia el sistema artificial que mantingui una conversa més indistinguible de la d'un humà. Avui dia, la comunitat científica considera que la intel·ligència artificial s'ha d'enfocar des d'una perspectiva diferent a la que es tenia en els anys cinquanta, però iniciatives com la de Loebner expressen l'impacte sociològic que continua tenint la IA en la societat actual.

Als anys vuitanta es van començar a desenvolupar les primeres aplicacions comercials de la IA, fonamentalment dirigides a problemes de producció, control de processos o comptabilitat. Amb aquestes aplicacions van aparèixer els primers sistemes experts, que permetien fer tasques de diagnòstic i presa de decisions a partir d'informació aportada per professionals experts. Entorn de 1990, IBM va construir l'ordinador d'escacs Deep Blue, capaç de plantar cara a un gran mestre d'escacs utilitzant algorismes de cerca i anàlisi que li permetien valorar centenars de milers de posicions per segon.

Més enllà de l'intent de dissenyar robots humanoides i sistemes que rivalitzin amb el cervell humà en funcionalitat i rendiment, l'interès avui dia és dissenyar i implementar sistemes que permetin analitzar grans quantitats de dades de manera ràpida i eficient. Actualment, cada persona genera i rep diàriament una gran quantitat d'informació no solament per mitjà dels canals clàssics (conversa, carta, televisió) sinó mitjançant nous mitjans que ens permeten contactar amb més persones i transmetre més dades en les comunicacions (Internet, fotografia digital, telefonia mòbil). Encara que el cervell humà és capaç de reconèixer patrons i establir relacions útils entre aquests de manera excepcionalment eficaç, és certament limitat quan la quantitat de dades resulta excessiva. Un fenomen similar ocorre en l'àmbit empresarial, en què cada dia és més necessari barrejar quantitats ingents d'informació per a poder pren-

HAL 9000

L'impacte social de la IA durant la dècada dels seixanta es posa de manifest en la pel·lícula *2001: A Space Odyssey*, dirigida el 1968 per Stanley Kubrick i basada en una novel·la homònima de ciència ficció d'Arthur C. Clarke. El protagonista principal de la pel·lícula és HAL 9000 (Heuristically programmed ALgorithmic computer), un ordinador dotat d'un avançat sistema d'intel·ligència artificial que és capaç de fer tasques com mantenir una conversa, reconeixement de veu, lectura de llavis, jugar a escacs i fins i tot manifestar un cert grau de sensibilitat artística.

Aplicacions de la IA

Actualment, la IA s'ha consolidat com una disciplina que permet dissenyar aplicacions de gran utilitat pràctica en nombrosos camps. Actualment, hi ha una enorme llista d'àmbits de coneixement en els quals s'utilitzen sistemes d'IA, entre els quals són d'especial rellevància la mineria de dades, el diagnòstic mèdic, la robòtica, la visió artificial, l'anàlisi de dades borsàries o la planificació i logística.

dre decisions. L'aplicació de tècniques d'IA als negocis ha donat lloc a àmbits d'implantació recent com la intel·ligència empresarial, *business intelligence*, o a la mineria de dades, *data mining*. En efecte, avui més que mai la informació està codificada en masses ingents de dades, de manera que en molts àmbits es fa necessari extreure la informació rellevant de grans conjunts de dades abans de procedir a una anàlisi detallada.

En resum, avui dia l'objectiu principal de la intel·ligència actual és el tractament i anàlisi de dades.

Algunes vegades ens interessarà caracteritzar les dades de manera simplificada per a poder fer una anàlisi en un espai de dimensió reduïda o per a visualitzar les dades de manera mes eficient. Per exemple, pot ser interessant saber quin subconjunt d'índexs borsaris internacionals són els més rellevants per a seguir la dinàmica d'un cert producte o mercat emergent. Altres vegades, l'objectiu serà identificar patrons en les dades per a poder classificar les observacions en diferents classes que resultin útils per a prendre decisions respecte d'un determinat problema. Un exemple d'aquest segon tipus d'aplicació seria l'anàlisi d'imatges mèdiques per a classificar pacients segons diferents malalties i així ajudar el metge en el seu diagnòstic. Finalment, en molts casos es fa necessari fer cerques entre una gran quantitat de dades o optimitzar una determinada funció de cost, per la qual cosa també serà necessari conèixer mètodes de cerca i optimització. En aquesta classe de problemes trobem, per exemple, el disseny dels horaris d'una estació de trens de manera que es minimitzi el temps d'espera i el nombre d'andanes utilitzades.

1.3. Àmbits d'aplicació de la intel·ligència artificial

Les aplicacions més freqüents de la intel·ligència artificial inclouen camps com la robòtica, l'anàlisi d'imatges o el tractament automàtic de textos. En *robòtica*, un dels camps d'investigació actual amb més projecció és el de l'*aprenentatge adaptatiu*, en el qual un sistema robotitzat explora diferents configuracions amb l'objectiu de fer un moviment complex (caminar, agafar un objecte, fer una trajectòria, jugar a golf, etc.). L'objectiu podria ser, per exemple, que un robot quadrúpede s'aixequi i camini de manera autònoma, de manera que segueixi un procés d'exploració i aprenentatge similar al que fa un nounat durant els primers mesos de vida. Un sistema d'IA en aquest cas s'encarregaria d'explorar diferents moviments de manera aleatòria, mesurant les variables de cada articulació i comprovant a cada moment el grau d'èxit aconseguit per cada seqüència d'accions (altura del centre de masses, desplaçament horizontal, fluctuacions en la posició vertical, velocitat de desplaçament, etc.).

El sistema d'IA modularia l'execució de les diferents accions, incrementant la probabilitat d'aquelles que presentin un millor rendiment i restringint les que no comportin una millora de la funció objectiu. Un àrea afí a la robòtica és la de les *interfícies cervell-computadora* (BCI, *brain-computer interfaces*), sistemes artificials que interactuen amb el sistema nerviós mitjançant senyals neurofisiològics amb l'objectiu assistir persones discapacitades durant l'execució de determinades tasques motores.

Dins del camp de la IA, una de les branques amb més projecció són els denominats *sistemes experts*, en els quals l'objectiu és dissenyar un sistema que permeti analitzar un conjunt de dades i fer tasques típicament associades a la figura d'un professional expert, com el diagnòstic, la detecció d'errors, la planificació o la presa de decisions. Les dades amb els quals treballa el sistema expert poden ser de naturalesa molt diversa.

En un sistema de diagnòstic clínic, per exemple, es pot partir d'imatges radioòptiques, d'una sèrie temporal de la freqüència del ritme cardíac d'un pacient o d'un conjunt de dades amb valors extrets d'anàlisis de sang o d'orina. La utilització de tots els senyals anteriors alhora constitueix un sistema de *fusió multimodal*, en el qual l'estat clínic del pacient es descriu des d'una perspectiva multiorgànica més completa. En els sistemes experts es combina informació extreta de dades amb el coneixement del sistema que aporta un expert especialitzat. Aquests sistemes es coneixen com a *sistemes basats en coneixement* (KBS, *knowledge-based systems*), i permeten integrar regles heurístiques i arbres de decisions elaborats per una comunitat d'experts durant anys de treball i experimentació. Aquestes regles no poden ser inferides directament de les dades observades, i són de gran utilitat en aplicacions sobre diagnòstic i presa de decisions.

Exemple

Un exemple d'aquest tipus de tècniques seria el protocol de diagnòstic que fa un metge especialista a partir d'un conjunt de proves, o els criteris que aplica un enginyer de camins per a validar la resistència mecànica d'un pont.

La informació aportada per humans experts és també necessària per a dissenyar sistemes artificials que juguin a jocs com els escacs o el go.

Go

El go és un joc de taula tradicional xinès que es practica en un tauler reticular i té per objectiu ocupar amb les fitxes una regió més gran que l'adversari. El desenvolupament de màquines que juguin a go amb humans a un alt nivell és un àmbit en el qual actualment es dediquen grans esforços des del camp de la IA. En aquest joc, el gran nombre de moviments possibles en cada jugada impedeix aplicar tècniques de cerca global, per la qual cosa els sistemes intel·ligents han d'incorporar informació sobre estratègies i tàctiques aportada per jugadors humans experts.

L'*anàlisi de textos* és un altre exemple interessant en el qual es desenvolupen nombrosos sistemes d'IA. Aspectes com la traducció automàtica de textos han evolucionat de manera sorprenent durant els últims anys gràcies a tècniques d'IA. Avui dia és fàcil trobar plataformes web gratuïtes que permeten traduir textos entre més de 50 llengües diferents. Aquests sistemes de traducció automàtica presenten resultats de notable qualitat, i tenen en compte aspectes semàntics i contextuels abans de proposar una traducció del text. L'èxit d'aquestes plataformes es deu en gran part a sistemes d'IA que utilitzen enormes quantitats d'informació textual aportada pels usuaris del servei. Sistemes similars permeten fer cerques intel·ligents en el Web, de manera que els vincles que s'ofereixen tinguin en compte les preferències estadístiques pròpies i de la resta d'usuaris, o enviar publicitat de manera selectiva a partir de la informació que apareix en el camp *assumpte* de la nostra bústia de correu electrònic.

Un altre exemple de sistema en l'àmbit de l'*enginyeria de processos* és un sistema de detecció d'errors en una planta complexa. Una planta industrial disposa de múltiples sensors distribuïts que permeten monitorar el procés de manera contínua. En la fase d'entrenament, el sistema aprèn un conjunt de patrons dinàmics dels sensors que corresponen a situacions en les quals es produeix un error en la planta. En el futur, quan es produeix un error, el sistema d'IA el detecta de manera automàtica i és capaç de *diagnosticar* l'origen de l'error comparant la resposta dels sensors amb les respistes característiques de cada error. En una tercera fase, el sistema pot fins i tot prendre decisions sobre quines accions cal fer per a resoldre el problema de la manera més ràpida i eficient. Les aplicacions industrials de la IA són un camp amb una gran projecció en el qual els sistemes van dirigits a millorar processos de fabricació, control de qualitat, logística o planificació de recursos.

Moltes vegades, els sistemes d'IA consten de dues fases, una primera fase d'*aprenentatge* i una segona de *predicció*. En la fase d'aprenentatge s'aporta un conjunt de dades representatiu d'aquelles situacions que es volen analitzar, de manera que el sistema IA aprèn les característiques fonamentals de les dades i és capaç de *generalitzar-ne* l'estructura. Aquesta generalització no és més que la construcció d'un model de les dades que permeti fer una predicción encertada a partir de noves observacions.

Reconeixement de cares

Considerem l'exemple d'un sistema automàtic per al reconeixement de cares. Al sistema se li proporcionen 100 imatges facials de 10 persones diferents, 10 cares per persona. Les imatges són preses en diferents condicions (diferent expressió facial, roba, il·luminació, exposició, fons, etc.), de manera que siguin representatives de les característiques facials de cada persona. El sistema identifica les característiques principals de cadascuna de les fotos i és capaç d'agrupar-les en un determinat nombre de grups (presumiblement 10, excepte si s'han inclòs bessons monozigòtics). Entre les característiques utilitzades poden aparèixer, entre altres, el color dels ulls, el gruix dels llavis o el perímetre del cap. En la fase de predicció, es parteix d'una imatge d'un dels 10 individus que no hagi estat inclosa en el conjunt de dades d'entrenament. El sistema calcula les característiques facials del nou individu, i ha de ser capaç d'identificar aquesta imatge com a pertanyent a un dels grups definits durant el procés d'aprenentatge (presumiblement al grup de l'individu

correcte, tret que la nova imatge no tingui amb prou feines trets comuns amb les 10 imatges d'entrenament). Si la nova imatge és identificada de manera correcta, direm que el sistema d'IA encerta en la predicció, mentre que quan el sistema assigna la imatge a un altre grup de manera errònia, es produeix un error de classificació.

Taula 2. Principals àmbits d'aplicació dels sistemes d'intel·ligència artificial

Àrea	Aplicacions
Medicina	Ajuda al diagnòstic Anàlisi d'imatges biomèdiques Processament de senyals fisiològics
Enginyeria	Organització de la producció Optimització de processos Càcul d'estructures Planificació i logística Diagnòstic d'errors Presa de decisions
Economia	Anàlisi financer i borsària Anàlisi de riscos Estimació de preus en productes derivats Mineria de dades Màrqueting i fidelització de clients
Biologia	Anàlisi d'estructures biològiques Genètica mèdica i molecular
Informàtica	Processament de llenguatge natural Criptografia Teoria de jocs Lingüística computacional
Robòtica i automàtica	Sistemes adaptatius de rehabilitació Interfícies cervell-computadora Sistemes de visió artificial Sistemes de navegació automàtica
Física i matemàtiques	Demostració automàtica de teoremes Anàlisi qualitativa sistemes de no lineals Caracterització de sistemes complexos

2. Recomanadors i agrupaments

Actualment la gran majoria d'instruments de mesura de qualsevol tipus de magnitud (des de magnituds físiques com longitud, temperatura, temps, fins a magnituds de comportament com patrons de cerca i navegació en el Web i preferències de compra en línia, passant per eines comunes com les càmeres digitals) són capaces d'abocar les mesures a algun format digital; d'aquesta manera totes aquestes dades poden estar fàcilment disponibles per al tractament.

En aquest moment el problema no és disposar de dades, ja que en tenim gran abundància; el repte és aconseguir extreure informació a partir de les dades, o sigui, donar-los un sentit i extreure'n conclusions útils. Aquesta tasca es coneix pel nom de **mineria de dades** (*data mining*).

Un dels principals reptes en el processament de dades és el d'integrar-les procedents de múltiples fonts, per a dotar així de diferents perspectives el conjunt d'aquestes dades, la qual cosa permet extreure informació més rica. Aquesta tendència es dóna en gairebé totes les àrees: recopilació de l'activitat dels usuaris en un lloc web; integració de sensors de temperatura, pressió atmosfèrica i vent en l'anàlisi meteorològica; ús de diferents dades financeres i borsàries en la previsió d'inversions, entre altres exemples.

La tasca d'integració de múltiples fonts de dades rep el nom de **filtratge col·laboratiu** (*collaborative filtering*).

De la mateixa manera que en les ciències experimentals és fonamental utilitzar un instrumental adequat i unes unitats consistentes, un aspecte clau en qualsevol tasca de processament de dades és l'ús de **mètriques**, o sigui, mesures de distància, adequades per al tipus de dades que s'està tractant.

Una de les aplicacions en les quals se centra aquest mòdul és la dels **recomanadors**. Un recomanador és un sistema que recull i analitza les preferències dels usuaris, generalment en algun lloc web (comerços, xarxes socials, llocs d'emissió o selecció de música o pel·lícules, etc.). La premissa bàsica dels reco-

Vegeu també

En el subapartat 2.1. d'aquest mòdul s'estudien algunes de les mètriques més habituals.

Vegeu també

En el subapartat 2.2. es descriuen algunes estratègies senzilles per a construir recomanadors.

manadors és que usuaris amb activitat o gustos similars continuaran comptant preferències en el futur. En recomanar a un usuari productes o activitats que altres usuaris amb gustos similars han triat prèviament el grau d'encert acostuma a ser més elevat que si les recomanacions es basen en tendències generals, sense personalitzar.

La tasca de trobar els usuaris més afins i utilitzar aquesta informació per a predir les seves preferències es pot inscriure en una tasca més general que rep el nom d'**agrupament** (*clustering*), i que consisteix a trobar la subdivisió òptima d'un conjunt de dades, de manera que les dades similars pertanyin al mateix grup.

Vegeu també

En el subapartat 2.3. es presenten els mètodes d'agrupament més importants.

2.1. Mètriques i mesures de similitud

Una **mètrica** és una funció que calcula la distància entre dos elements i que, per tant s'utilitza per a mesurar com són de diferents. Hi ha diverses maneres de mesurar la distància entre dos elements, i triar la mètrica adequada per a cada problema és un pas crucial per a obtenir bons resultats en qualsevol aplicació de mineria de dades.

Sovint s'utilitzen funcions que mesuren la similitud entre dos elements en lloc de la seva distància.

En aquest subapartat utilitzarem la **distància euclidiana** i estudiarem una mesura de similitud habitual, la **correlació de Pearson**.

Finalment, una manera habitual i segura* de convertir una funció de distància d en una funció de similitud s és la següent:

$$s(P, Q) = \frac{1}{1 + d(P, Q)} \quad (1)$$

Vegeu també

Les distàncies i similituds s'utilitzen en gran quantitat de mètodes, per la qual cosa en aquest mòdul es presenten altres mètriques com la distància de Hamming en el subapartat 4.3.1. i la informació mútua en el subapartat 3.1.3.

2.1.1. Exemple d'aplicació

En un lloc web de visualització de pel·lícules a la carta es recull la valoració de cada usuari sobre les pel·lícules que va veient, amb l'objectiu de poder proposar als usuaris les pel·lícules que més s'adaptin als seus gustos. Després de veure una pel·lícula, un usuari ha de donar una valoració entre 1 i 5, en què les valoracions més baixes corresponen a pel·lícules que no han agradat a l'usuari, i les més altes a les que li han agradat més. Es volen descobrir similituds entre usuaris de manera que a cada usuari se li proposin les pel·lícules que més han agradat als usuaris amb gustos més semblants al seu.

* Segura perquè no incorre en divisions per zero si la distància és zero.

Per a posar en pràctica aquestes proves s'utilitzaran els conjunts de dades disponibles a <http://www.grouplens.org/node/73>, concretament el conjunt de 100k

valoracions (fitxer *ml-data_0.zip*), que conté 100.000 valoracions (de l'1 al 5) de 1.682 pel·lícules fetes per 943 usuaris. Si bé el conjunt de dades conté 23 fitxers, en aquest mòdul només s'utilitzarà el fitxer *u.data*, que és el que conté les valoracions dels usuaris. En la taula 3 es mostra un exemple merament il·lustratiu de valoracions.

Taula 3. Valoracions de 8 pel·lícules (exemple)

IdUsuari/IdPel·lícula	1	2	3	4	5	6	7	8
1		3	1		4	3	5	
2	4	1	3		5			2
3	2	1		5				1
4	3		2			5		4

En el fitxer *u.data* cada valoració apareix en una fila, i les columnes són *IdUsuari* *IdPel·lícula* *Valoració* *Data*.

Les dades es presenten en forma matricial per lleigibilitat i concisió.

El fitxer *u.item* conté informació sobre cada pel·lícula, i el fitxer *u.user* conté informació genèrica sobre cada usuari (edat, sexe, etc.). Com es pot observar, els usuaris no necessàriament han de valorar totes les pel·lícules, sinó només les que han vist. La informació de la taula es pot expressar en Python mitjançant un diccionari en el qual cada identificador d'usuari és una clau, i el valor associat a aquesta clau és un altre diccionari que associa cada identificador de pel·lícula a la puntuació assignada per aquest usuari. El fragment de codi 2.1 mostra la representació corresponent a la taula 3.

Codi 2.1: Diccionari de valoració de pel·lícules

```
1 valoracions = {1: {2:3,3:1,5:4,6:3,7:5}, 2: {1:4,2:1,3:3,5:5,8:2},
2   3: {1:2,2:1,4:5,8:1}, 4: {1:3,3:2,6:5,8:4}}
```

2.1.2. Distància euclidiana

La distància euclidiana de dos punts $P = (p_1, p_2, \dots, p_n)$ i $Q = (q_1, q_2, \dots, q_n)$ en l'espai de dimensió n \mathbb{R}^n està determinada per la fórmula:

$$d(P, Q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \quad (2)$$

La distància euclidiana no és més que la generalització a n dimensions del teorema de Pitàgores. Si les distàncies en si no són importants, sinó només la comparació, sovint s'utilitza la **distància euclidiana quadrada**, és a dir, sense l'arrel quadrada, ja que la comparació entre distàncies euclidianes quadrades dóna els mateixos resultats que entre les distàncies euclidianes i pot resultar molt més ràpida de calcular, ja que l'arrel quadrada és una operació computacionalment costosa.

La funció que calcula la distància euclidiana entre les valoracions de dos usuaris (segons el format vist en el fragment de programa 2.1) en Python es mostra en el fragment de programa 2.2, i també la funció necessària per a convertir la distància en una similitud.

Codi 2.2: Distància euclidiana entre dos diccionaris

```

1 from math import sqrt
2
3 def distEuclidiana(dic1, dic2):
4     # Calcular la suma de quadrats dels elements comuns
5     # als dos diccionaris
6     suma2 = sum([pow(dic1 [elem]-dic2 [elem], 2)
7                 for elem in dic1 if elem in dic2])
8     return sqrt(suma2)
9
10 def similEuclidiana(dic1, dic2):
11     return 1/(1+distEuclidiana(dic1, dic2))

```

En la taula 4 es mostren les similituds entre les valoracions dels usuaris utilitzant la distància euclidiana. Com es pot comprovar, $s(P,P) = 1$ (ja que $d(P,P) = 0$) i $s(P,Q) = s(P,Q)$, o sigui, la matriu de similituds és simètrica.

Taula 4. Similitud euclidiana de les valoracions

ID usuari	1	2	3	4
1	1,0	0,25	0,33	0,31
2	0,25	1,0	0,31	0,29
3	0,33	0,31	1,0	0,24
4	0,31	0,29	0,24	1,0

Suposem que l'usuari 4 vol que li recomanin alguna pel·lícula; el sistema, de moment, li pot suggerir que vegi les pel·lícules que han agrat a l'usuari 1 (i que no ha vist el 4), ja que és l'usuari amb més similitud; en l'exemple, l'usuari 4 podria triar les pel·lícules 5 i 7, que són les que més han agrat a l'usuari 1 i encara no ha vist el 4.

Una limitació de la similitud (i la distància) euclidiana és que és molt sensible a l'escala: si un usuari tendeix a puntuar les pel·lícules que li agraden amb un 4 i un altre amb un 5, hi haurà una certa distància entre ells, especialment si el nombre de valoracions és alt, encara que en el fons tots dos usuaris comparteixen gustos malgrat utilitzar les puntuacions de maneres diferents.

A més, com més dimensions (valoracions, en aquest cas), la distància euclidiana tendeix a ser més gran, la qual cosa pot distorsionar els resultats. Per exemple, si dos usuaris comparteixen 2 valoracions i hi ha una diferència d'1 entre cadascuna, la distància serà $d = \sqrt{1^2 + 1^2} = 1,41$; no obstant això, si comparteixen 5 valoracions amb una diferència d'1, la distància serà $d = \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2} = 2,24$; una distància bastant més gran per a unes valoracions en aparença similars. Com més valoracions comparteixin dos usuaris (encara que no siguin gaire diferents), més allunyats estaran, la qual cosa sembla contrària a la lògica.

Utilitat de la distància euclidiana

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.

2.1.3. Correlació de Pearson

El coeficient de correlació de Pearson és una mesura de similitud entre dues variables que resol els problemes de la similitud euclidiana. Es tracta d'una mesura de com les dues variables, una enfront d'una altra, s'organitzen entorn d'una línia recta (línia de millor ajust), tal com es pot veure en la figura 1. Com més similars són les valoracions de dos usuaris, més s'assemblarà la seva recta a la recta $y = x$, ja que les valoracions seran de la forma (1,1), (3,3), (4,4), etc.

Figura 1. Correlació entre les valoracions els usuaris 1 i 2 de la taula 3

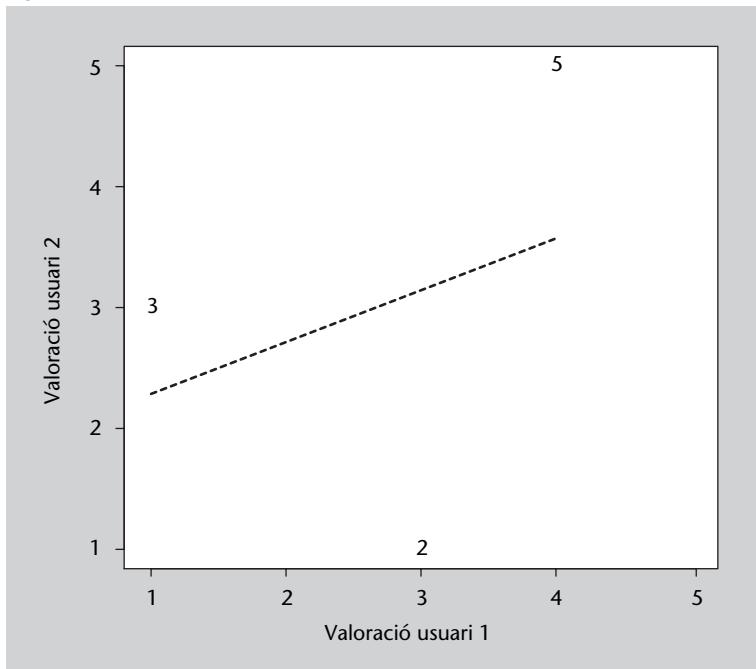


Figura 1

Diagrama de dispersió (*scatter plot*) de les valoracions de dos usuaris: es representa la valoració de cada pel·lícula en comú prenent l'eix x com la valoració d'un usuari, i l'eix y com la valoració de l'altre usuari. Els nombres en el diagrama corresponen als identificadors de les pel·lícules comunes.

El coeficient de correlació de Pearson (en aquest subapartat, simplement *correlació*) està relacionat amb el pendent de la recta representada en la figura 1, i pot prendre un valor en el rang $[-1,1]$. Si el valor és 1 indica que les dues variables estan perfectament relacionades; si és 0, no hi ha relació lineal entre aquests*; si és negatiu és que hi ha una correlació negativa, en aquest cas les valoracions d'un usuari són oposades a les de l'altre.

* El coeficient de correlació de Pearson només mesura relacions lineals; encara que valgui 0, hi pot haver relacions no lineals entre les dues variables.

El càlcul del coeficient de correlació de Pearson sobre dues mostres de dades alineades (valoracions d'usuaris, en el nostre cas) x_i i y_i està determinat per la fórmula:

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (3)$$

en què \bar{x} és la mitjana dels valors de x i \bar{y} la mitjana dels valors de y .

Noteu que per a efectuar el càlcul les dades han d'estar alineades: en el nostre cas, només s'han de prendre les valoracions comunes als dos usuaris. També cal preveure que el denominador pugui valdre zero. Amb totes aquestes consideracions, en el codi 2.3 es mostra la funció en Python que calcula el coeficient de correlació de Pearson de dos diccionaris de valoracions.

Codi 2.3: Coeficient de Pearson entre dos diccionaris

```

1 def coefPearson(dic1, dic2):
2     # Obtenir els elements comuns als dos diccionaris
3     comuns = [x for x in dic1 if x in dic2]
4     nComuns = float(len(comuns))
5
6     # Si no hi ha elements comuns, es retorna zero; si no
7     # es calcula el coeficient
8     if nComuns==0:
9         return 0
10
11    # Càlcul de les mides de cada diccionari
12    mitjana1 = sum([dic1[x] for x in comuns])/nComuns
13    mitjana2 = sum([dic2[x] for x in comuns])/nComuns
14
15    # Càlcul del numerador i del denominador
16    num = sum([(dic1[x]-mitjana1)*(dic2[x]-mitjana2) for x in comuns])
17    den1 = sqrt(sum([pow(dic1[x]-mitjana1, 2) for x in comuns]))
18    den2 = sqrt(sum([pow(dic2[x]-mitjana2, 2) for x in comuns]))
19    den = den1*den2
20
21    # Càlcul del coeficient si és possible, o retorna 0
22    if den==0:
23        return 0
24
25    return num/den

```

Com es dedueix de la definició, el coeficient de correlació de Pearson és simètric i val 1 en calcular-lo respecte a la mateixa variable. En la taula 5 es poden veure els valors corresponents a les valoracions dels usuaris de la taula 3. En aquest exemple la correlació entre els usuaris 1 i 3 és 0 perquè només tenen una pel·lícula en comú, amb la qual cosa la recta d'ajust no es pot definir.

Taula 5. Coeficient de Pearson de les valoracions

ID usuari	1	2	3	4
1	1,0	0,33	0,0	1,0
2	0,33	1,0	0,95	-0,5
3	0,0	0,95	1,0	-1,0
4	1,0	-0,5	-1,0	1,0

També s'observa que els usuaris 1 i 4 tenen una correlació d'1,0; això és a causa que, independentment de factors d'escala, la seva tendència a valorar les pel·lícules és igual. Noteu la diferència amb la similitud euclidiana mostrada en la taula 4, si bé l'usuari més similar al 4 continua essent l'1 i per tant les

pel·lícules que podria triar serien les mateixes. D'altra banda, els usuaris 3 i 4 fan valoracions contràries, per la qual cosa el seu coeficient és -1,0. No obstant això, seria convenient disposar de més dades per a obtenir mesures més realistes.

El coeficient de correlació de Pearson resulta útil com a mesura de similitud perquè és independent dels desplaçaments i escales dels valors estudiats; per a aconseguir aquestes propietats utilitzant la similitud euclidiana seria necessari normalitzar les dades abans de l'anàlisi. Un dels principals inconvenients en l'ús en filtratge col·laboratiu és que requereix que hi hagi almenys dos valors comuns per a poder donar un resultat significatiu; això en limita l'aplicació en aquells casos en què volem suggerir productes similars a un client que ha fet una única elecció.

2.1.4. Processament de dades reals

Fins ara s'han utilitzat les funcions de similitud sobre dades de prova introduïdes manualment; no obstant això, és interessant provar-les amb un volum de dades apreciable per a poder-ne valorar millor el comportament.

En el fragment de codi 2.4 es llegeixen les dades del fitxer *u.data* de la base de dades de valoracions de pel·lícules MovieLens utilitzada en aquest subapartat, i es produeix com a resultat un diccionari amb la mateixa estructura que el que es dóna com a exemple en el codi de programa 2.1.

Per a utilitzar-lo n'hi ha prou d'aplicar alguna de les funcions de similitud descrites anteriorment a qualsevol parell d'usuaris (claus del diccionari principal), o bé escriure un programa que generi una taula o diccionari amb totes les correlacions.

Codi 2.4: Funció que llegeix un fitxer de valoracions de MovieLens

```

1 def llegeixValoracions(nomFitx="u.data"):
2     linies = [(l.strip()).split("\t")
3             for l in (open(nomFitx).readlines())
4             diccio = {int(l[0]) : {} for l in linies}
5
6             for l in linies:
7                 diccio[int(l[0])][int(l[1])] = int(l[2])
8             return diccio

```

2.1.5. Conclusions

Les principals limitacions dels procediments explícats en aquest subapartat són dues: primera, que no permeten suggerir productes directament, sinó només usuaris afins; aquesta limitació es resoldrà en el subapartat següent. La segona limitació és que és necessari emmagatzemar en memòria totes les va-

loracions dels usuaris i recalcular-les totes cada vegada que s'incorpora una nova valoració o usuari, la qual cosa pot representar un cost computacional alt en aplicacions grans; vist d'una altra manera, no es genera cap abstracció o model de les dades que permeti treballar amb una versió reduïda de les dades. Aquesta limitació es resoldrà en el subapartat 2.3., que tracta sobre els algorismes d'agrupament.

2.2. Recomanadors basats en memòria

En aquest subapartat es millorarà el sistema de recomanació de pel·lícules vist en el subapartat 2.1. perquè, en lloc de suggerir usuaris amb gustos similars, suggereixi directament les pel·lícules que poden ser més interessants per a l'usuari.

2.2.1. Conceptes generals

Es parla de recomanadors **basats en memòria** perquè requereixen mantenir totes les dades disponibles per a poder fer una recomanació; a més, cada vegada que s'introdueix una nova dada o que se sol·licita una recomanació aquests recomanadors han d'efectuar les seves operacions sobre el conjunt complet de dades, la qual cosa provoca que siguin relativament ineficients, especialment en aplicacions amb volums de dades grans. No obstant això, tenen interès perquè són molt senzills conceptualment i la programació és relativament ràpida.

Aquests recomanadors estan estretament relacionats amb els classificadors basats en distàncies estudiats en el subapartat 4.3., ja que fan servir una metodologia similar (per exemple, buscar l'element més proper al que s'està avaluant); la diferència es troba en l'ús que es fa d'aquesta informació, ja que en el cas dels classificadors es pren un nou element i es decideix a quina classe ha de pertànyer, mentre que en el cas dels recomanadors es pren un element i se suggereixen els elements més semblants, sense necessitat de fer una classificació.

En la pràctica es tracta de generar una valoració (rànquing) de totes les dades o preferències registrades i que tingui en compte les particularitats del sol·licitant (gustos de l'usuari, etc.).

2.2.2. Aproximacions simples

La manera més senzilla de generar una valoració global d'un conjunt d'elements és amitjanar-ne grau de valoració. En l'exemple de les pel·lícules, si una pel·lícula té quatre valoracions (3, 5, 4, 3), la valoració mitjana serà 3,75; les pel·lícules amb més valoració global seran les suggerides a l'usuari. En el co-

di 2.5 es mostra la funció que retorna una llista ordenada (de més a menys valoració) de pel·lícules, a partir del diccionari de valoracions dels usuaris.

Codi 2.5: Funció de recomanació mitjana global de pel·lícules

```

1 # Genera una llista ordenada de valoracions globals a partir
2 # d'un diccionari de valoracions d'usuaris, de la forma
3 # [(idPel·licula , valoracioGlobal)]
4 def llistaValoracionsSimple(diccio):
5     # Diccionari auxiliar {idPel·licula: [valoracions]}
6     aux = {}
7     for valorUsuari in dicció.values():
8         for idPel·li in valorUsuari:
9             if not aux.has_key(idPel·li):
10                 aux[idPel·li] = []
11                 aux[idPel·li].append(valorUsuari[idPel·li])
12
13     # Càcul i ordenació de les valoracions globals
14     mitjana = lambda x: sum(x)/float(len(x))
15     result = [(p, mitjana(aux[p])) for p in aux]
16     result.sort(key = lambda x: x[1], reverse=True)
17     return result

```

Òbviament és possible afegir un *if* per a ometre de la llista les pel·lícules que ja ha vist l'usuari, encara que no s'ha inclòs aquesta operació per simplificar el codi font mostrat.

La principal limitació d'aquest mètode és que no té en compte les preferències individuals de cada usuari. No obstant això, aquesta estratègia pot ser útil amb usuaris nous, dels quals no es coneixen els gustos. Un avantatge és que no depèn de cap mètrica per a efectuar les recomanacions.

Una estratègia senzilla que té en compte les preferències de l'usuari consisteix a seleccionar les pel·lícules favorites de l'usuari més semblant, que era el que es deixava fer manualment en el subapartat 2.1. després de suggerir l'usuari més proper. El desavantatge d'aquesta estratègia és que està limitada a les pel·lícules que ha vist l'usuari més proper. Si no ha vist una pel·lícula que molts altres usuaris relativament afins han vist i valorat positivament, el sistema mai no suggerirà aquesta pel·lícula a pesar que es podria tractar d'una bona recomanació. Aquesta estratègia és similar al mètode de classificació del veí més proper (*1-Nearest-Neighbour*), que s'estudia en el subapartat 4.3.1.

2.2.3. Recomanació ponderada

És possible utilitzar un mètode que reuneix els avantatges dels dos mètodes de recomanació simples que s'acaben d'exposar. La idea és senzilla: es tracta d'obtenir una valoració global del conjunt d'elements (productes) però en lloc de calcular la mitjana aritmètica de les valoracions, cal calcular la mitjana de les valoracions ponderada per l'afinitat de l'usuari corresponent. D'aquesta manera, la valoració global dels productes està personalitzada per a cada usuari

i, per tant, hauria de resultar més útil i encertada. En el programa 2.6 es mostra una funció que calcula la valoració mitjana de totes les pel·lícules ponderada per l'afinitat dels usuaris amb l'usuari actual. $vp_j(x)$ és la valoració ponderada d'una pel·lícula x per a l'usuari j , i està determinada per l'expressió:

$$vp_j(x) = \begin{cases} \frac{\sum_{i \in V(x)} (s(i,j)v_i(x))}{\sum_{i \in V(x)} s(i,j)} & \text{si } |V(x)| > 0 \\ 0 & \text{si } |V(x)| = 0 \end{cases} \quad (4)$$

en què $v_i(x)$ és la valoració de la pel·lícula x de l'usuari i i $s(i,j)$ és la similitud entre els usuaris i i j . Finalment, $V(x)$ és el conjunt d'usuaris que han valorat la pel·lícula x .

Codi 2.6: Funció de recomanació mitjana ponderada de pel·lícules

```

1 # Genera una llista ordenada de valoracions ponderades a partir
2 # d'un diccionari de valoracions d'usuaris i un número d'usuari.
3 # Es pot triar la funció de similitud entre usuaris.
4 def valoracioPonderada(diccio, usuari, similitud = coefPearson):
5     # En primer lloc, es genera un diccionari amb les similituds
6     # del nostre usuari amb tots els altres.
7     # Aquest diccionari es podria emmagatzemar per a evitar recalcular-lo.
8     simils = {x: similitud(diccio[usuari], diccio[x])
9             for x in diccio if x != usuari}
10
11    # Diccionari auxiliar {idPel·licula: [valoració*similitud
12    # usuaris]} i {idPel·licula: [similitud usuaris]} (numerador i
13    # denominador de la valoració ponderada)
14    numerador = {}
15    denominador = {}
16
17    # Es recorre el diccionari de valoracions i s'omplen els
18    # diccionaris auxiliars amb els valors trobats
19    for idUsuari in simils:
20        for idPel·li in diccio[idUsuari]:
21            if not numerador.has_key(idPel·li):
22                numerador[idPel·li] = []
23                denominador[idPel·li] = []
24            s = simils[idUsuari]
25            numerador[idPel·li].append(diccio[idUsuari][idPel·li]*s)
26            denominador[idPel·li].append(s)
27
28    # Es calculen i ordenen les valoracions ponderades
29    result = []
30    for idPel·li in numerador:
31        s1 = sum(numerador[idPel·li])
32        s2 = sum(denominador[idPel·li])
33        if s2 == 0:
34            mitjana = 0.0
35        else:
36            mitjana = s1/s2
37            result.append((idPel·li, mitjana))
38
39    result.sort(key = lambda x: x[1], reverse=True)
40    return result

```

2.2.4. Conclusions

Els recomanadors vists en aquest subapartat permeten millorar d'una manera molt senzilla llocs web i altres aplicacions similars per a sintonitzar millor

amb els usuaris i suggerir-los productes que els puguin interessar, i millorar així tant la seva satisfacció amb l'aplicació com els èxits potencials de venda o accés.

Les limitacions dels mètodes vistos fins ara són que, com s'ha explicat, són mètodes basats en memòria: requereixen emmagatzemar i processar totes les dades cada vegada que es fa una consulta. D'aquesta manera, una operació senzilla no resulta excessivament costosa, però com s'ha de repetir completament en cada consulta pot donar lloc a una càrrega computacional i de memòria desmesurada en aplicacions mitjanes i grans.

Una altra limitació fonamental és que els mètodes de recomanació que s'acaben d'estudiar no abstreuen les dades de cap manera, és a dir, no proporcionen informació global sobre les dades de què es disposa, amb la qual cosa l'ús està limitat a produir recomanacions de productes, però no es poden utilitzar per a abordar estudis més complexos sobre els tipus d'usuaris o productes de què es disposa, estudis que són essencials per a analitzar el funcionament de l'aplicació i dissenyar-ne el futur. A continuació s'estudiaran mètodes que sí que abstreuen i produeixen informació d'alt nivell a partir de les dades disponibles.

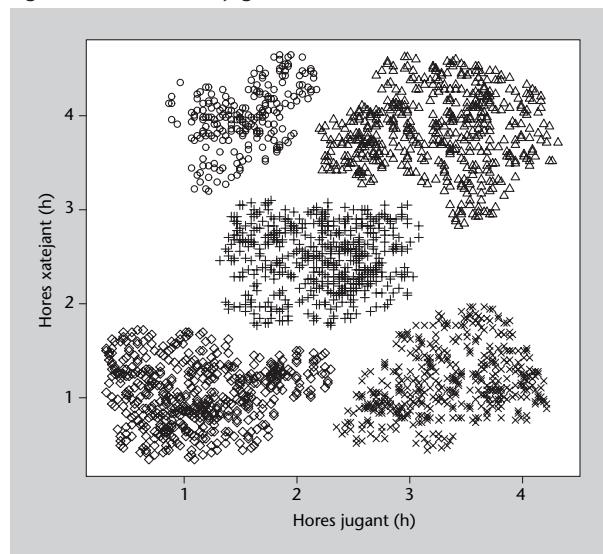
2.3. Algorismes d'agrupament (*clustering*)

2.3.1. Exemple d'aplicació

Una empresa de jocs en línia està analitzant el comportament dels seus clients amb la finalitat d'ofrir-los els productes i característiques més adequats als seus interessos. En aquest estudi es mesuren dues característiques: nombre d'hores diàries jugant i nombre d'hores diàries xerrant (xatejant) amb altres jugadors, tots dos en valor de mitjana per a cada jugador. Els objectius són dos: el primer, determinar quins tipus o classes de jugadors hi ha, segons la seva activitat; el segon, classificar els nous jugadors en alguna d'aquestes classes per a poder oferir-los les condicions i productes més adequats. El segon objectiu es tracta extensament en l'apartat 3, dedicat a la classificació, mentre que en aquest subapartat ens centrarem en com podem organitzar un conjunt de dades extens en unes quantes classes o grups, desconeguts *a priori*.

En la figura 2 es mostren les dades recollides de 2.084 jugadors. Es tracta de dades sintètiques (generades artificialment) amb el propòsit que il·lustrin més clarament els mètodes que s'exposaran a continuació. Concretament, es poden observar cinc classes o grups clarament diferenciats, cadascun marcat amb un símbol diferent. En les dades reals no se sol observar una separació tan clara, i per tant l'execució dels mètodes d'agrupament no produeix resultats tan definits.

Figura 2. Activitat dels jugadors en línia

**Figura 2**

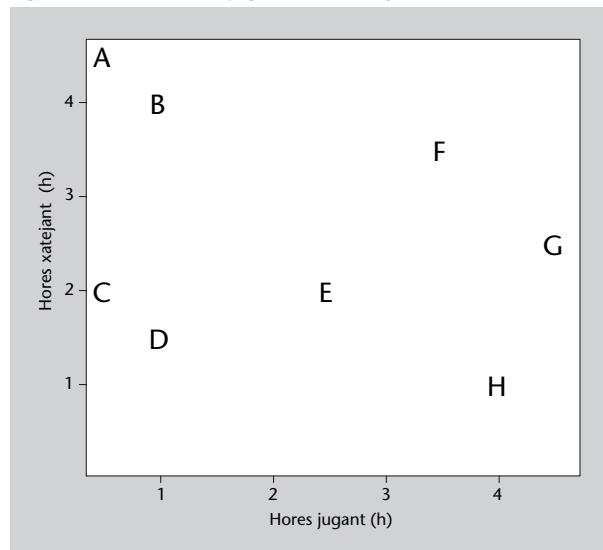
Les dues variables utilitzen les mateixes unitats (hores) i magnituds similars i, per tant, es poden utilitzar directament per a agrupar; en un cas més general, en el qual les variables tinguin magnituds diferents, sol ser necessari normalitzar-les perquè la magnitud s'equipari i no tinguin més influència unes que altres (tret que es pretengui justament això per la naturalesa del problema).

De fet, per a mostrar l'execució pas per pas d'alguns algorismes d'agrupament és necessari utilitzar moltes menys dades, que en aquest cas són les que es mostren en la taula 6 i en la figura 3.

Taula 6. Jugadors d'exemple

Jugador	Hores joc	Hores xat
A	0,5	4,5
B	1	4
C	0,5	2
D	1	1,5
E	2,5	2
F	3,5	3,5
G	4,5	2,5
H	4	1

Figura 3. Activitat de 8 jugadors d'exemple



2.3.2. Conceptes generals

La tasca d'agrupament de 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 mètode d'agrupament mateix el que ha de descobrir les noves classes o grups a partir de les dades rebudes.

Si bé els algorismes d'agrupament s'han introduït en el subapartat anterior com a eina per a crear recomanadors, les seves aplicacions van molt més allà i s'empren en nombrosos àmbits, com per exemple:

- Processament d'imatge, per a separar unes zones d'altres (típicament amb imatges de satèl·lit).
- Agrupament de gens relacionats amb una determinada característica o malaltia.
- Agrupament automàtic de textos per temes.
- Definir tipus de clients i orientar les estratègies comercials en funció d'aquests grups.
- En general, definir grups en conjunts de dades dels quals no es coneix una subdivisió prèvia: astronomia, física, química, biologia, medicina, farmacologia, economia, sociologia, psicologia, etc.

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, com es podrà observar en aquest subapartat, que comparteix algunes tècniques amb el subapartat 4.3.

Els algorismes d'agrupament es poden classificar en dos tipus: els **jeràrquics** amb progressius, és a dir, que van formant grups progressivament, i s'estudiaran en primer lloc; els **particionals** només calculen una partició de les dades: la resta d'algorismes estudiats en aquest subapartat pertanyen a aquesta categoria.

Utilitzant algorismes d'agrupament és possible construir recomanadors **basats en models**, anomenats així perquè, a diferència dels recomanadors basats en dades, no necessiten emmagatzemar totes les dades de què es disposa, sinó que produeixen una abstracció de les dades dividint-les en grups; per a produir una recomanació només és necessari associar un usuari o producte a un grup existent, per la qual cosa la informació per emmagatzemar es redueix a la descripció dels grups obtinguts prèviament. D'aquesta manera la generació d'una recomanació se simplifica notablement, i per tant els recursos requerits per a generar-la.

Més enllà dels recomanadors, una de les característiques més importants dels algorismes d'agrupament és que permeten organitzar dades que en principi no sap o pot classificar, i eviten 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 bé 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.

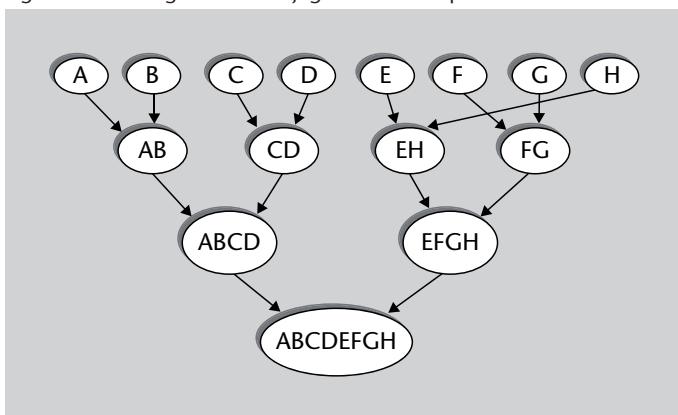
2.3.3. Agrupament jeràrquic. Dendrogrames

Hi ha dos tipus d'algorismes d'agrupament jeràrquics. L'algorisme **aglomeratiu** parteix d'una fragmentació completa de les dades (cada dada té el seu grup propi) i fusiona grups progressivament fins a assolir la situació contrària: totes les dades estan reunides en un únic grup. L'algorisme **divisiu**, per la seva banda, procedeix de la manera oposada: parteix d'un únic grup que conté totes les dades i el va dividint progressivament fins a tenir un grup per a cada dada.

El **dendrograma*** és un diagrama que mostra les agrupacions successives que genera (o desfà) un algorisme d'agrupament jeràrquic. En la figura 4 es pot veure el dendrograma resultant d'aplicar agrupament aglomeratiu les dades mostrades en la taula 6.

*O diagrama d'arbre, del grec *dendron*, 'arbre', i *gramma*, 'dibuix'.

Figura 4. Dendrograma dels 8 jugadors d'exemple de la taula 6



En els algorismes jeràrquics d'agrupament es parteix de l'estat inicial i s'aplica un criteri per a decidir quins grups cal unir o separar en cada pas, fins que s'assoleix l'estat final. Si bé conceptualment tots dos tipus d'algorismes jeràrquics (aglomeratiu i divisiu) són equivalents, en la pràctica l'algorisme aglomeratiu és més senzill de dissenyar per la raó que només hi ha una manera d'unir dos conjunts, però hi ha moltes maneres de dividir un conjunt de més de dos elements.

Criteris d'enllaç

En cada pas d'un algorisme d'agrupament jeràrquic cal decidir quins grups cal unir (o dividir). Per a això cal determinar quins grups són més propers, o bé quin grup està menys cohesionat. Suposem que estem programant un algorisme aglomeratiu; la distància entre dos grups es pot determinar segons diferents expressions, que reben el nom de **criteris d'enllaç**. Alguns dels criteris més utilitzats per a mesurar la distància entre dos grups A i B són els següents:

- Distància màxima entre elements dels grups (enllaç complet):

$$\max\{d(x,y) : x \in A, y \in B\}$$

- Distància mínima entre elements dels grups (enllaç simple):

$$\min\{d(x,y) : x \in A, y \in B\}$$

- Distància mitjana entre elements dels grups (enllaç mitjà):

$$\frac{1}{|A||B|} \sum_{x \in A} \sum_{y \in B} d(x,y)$$

Un desavantatge del criteri d'enllaç simple és que pot provocar que un únic element forci la unió de tot el seu conjunt amb un altre conjunt que, d'altra banda, no sigui especialment proper. Per aquesta raó en general es preferiran altres criteris (complet, mitjà o altres).

Codi de l'algorisme aglomeratiu

En el programa següent es defineixen les funcions necessàries per a calcular l'agrupament aglomeratiu sobre conjunt de dades utilitzant distància euclidiana (es podria utilitzar una altra distància) i criteris d'enllaç complet o mínim. La

funció *fusionaGrups* pren un agrupament, selecciona els dos grups més propers segons el criteri indicat i els fusiona; la funció *agrupamentAglomeratiu* inicialment genera un grup per a cada dada i els va fusionant fins que només queda un grup; en cada fusió mostra l'agrupament resultant. Finalment s'inclou una funció *llegeixJugadors*, que llegeix un fitxer amb les dades dels jugadors en format "horesJoc horesXat classe" (s'inclou la classe per a avaluar el rendiment del programa).

Codi 2.7: Agrupament aglomeratiu

```

1  from math import sqrt
2
3  # Calcula la distància euclidiana entre dos vectors/tuples
4  def distEuclidiana(v1, v2):
5      return sqrt(sum(pow(x-y,2) for x,y in zip(v1,v2)))
6
7
8  # Calcula l'enllaç complet (màxim) entre dos grups
9  # (distància màxima entre dos punts de cada grup)
10 def enllaçComplet(punts, g1, g2, dist = distEuclidiana):
11     # Busca el màxim en les combinacions de punts
12     maxima = 0.0
13     for p1 in g1:
14         for p2 in g2:
15             d = dist(punts[p1], punts[p2])
16             if d > maxima:
17                 maxima = d
18     return maxima
19
20
21 # Calcula l'enllaç simple (mínim) entre dos grups
22 # (distància mínima entre dos punts de cada grup)
23 def enllaçSimple(punts, g1, g2, dist = distEuclidiana):
24     # Busca el mínim en les combinacions de punts
25     minima = float("inf")
26     for p1 in g1:
27         for p2 in g2:
28             d = dist(punts[p1], punts[p2])
29             if d < minima:
30                 minima = d
31     return minima
32
33 # Donat un conjunt de punts i un agrupament, fusiona
34 # els dos grups més propers amb el criteri indicat.
35 # "grups" ha de contenir almenys dos grups, i torna
36 # modificat, amb els grups triats fusionats.
37 def fusionaGrups(punts, grups, criteri=enllaçComplet,
38                  dist=distEuclidiana):
39     if len(grups) < 1: return
40
41     # Busca el parell de grups més adequats (valor mínim
42     # del criteri utilitzat).
43     minim = float("inf")
44     noms = grups.keys()
45     for i in range(len(noms)-1):
46         for j in range(i+1, len(noms)):
47             d = criteri(punts, grups[noms[i]], grups[noms[j]], dist)
48             if d < minim:
49                 minim = d
50                 candidat = (noms[i], noms[j])
51
52     # El nom del nou grup serà el més baix dels dos
53     nomGrup = min(candidat)
54     grupEsborrar = max(candidat)
55
56     # Fusiona els dos grups: afegeix els elements a un
57     # dels grups i elimina l'altre del diccionari "grups".
58     grups[nomGrup].extend(grups[grupEsborrar])
59

```

```

60 def(grups [grupEsborrar]):
61
62
63 # Agrupament jeràrquic aglomeratiu: fusiona grups
64 # fins a obtenir un únic grup
65 def agrupamentAglomeratiu(punts, criteri=enllaçComplet,
66                         dist=distEuclidiana):
67     # Generació de l'agrupament inicial (cada punt un grup)
68     grups = {x:[x] for x in punts}
69     print(grups)
70
71     # Fusió de grups fins a aconseguir un únic grup
72     while len(grups) > 1:
73         fusionaGrups(punts, grups, criteri, dist)
74         print(grups)
75
76 # Llegeix un fitxer amb les dades dels jugadors, format
77 # "horesJoc horesXat classe" (la classe s'ignora)
78 def llegeixJugadors(nomFitx="Jugadors.txt"):
79     línies = [(l.strip()).split("\t")
80               for l in (open(nomFitx).readlines())
81     # S'assigna un id=0,1,2,... a cada jugador
82     dicció = {}
83     for i in range(len(línies)):
84         dicció[i] = (float(línies[i][0]), float(línies[i][1]))
85     return diccio
86
87
88 # Diccionari de punts d'exemple amb les seves coordenades
89 punts = { 'A':(0.5,4.5), 'B':(1,4), 'C':(0.5,2), 'D':(1,1.5),
90          'E':(2.5,2), 'F':(3.5,3.5), 'G':(4.5,2.5), 'H':(4,1)}

```

Una possible optimització del programa consistiria a emmagatzemar les distàncies entre grups per no haver de recalcular-les totes en cada fusió. En el programa d'exemple no s'ha fet així per no complicar-ho per qüestions de simple eficiència.

Conclusions

El principal interès dels algorismes jeràrquics és que generen diferents graus d'agrupament, i la seva evolució, la qual cosa pot ser tan útil com l'obtenció d'un agrupament definitiu. A més, són senzills conceptualment i des del punt de vista de la programació. D'altra banda, els algorismes jeràrquics d'agrupament actuen de manera **voraç** (*greedy*), ja que en cada pas prenen la millor decisió en aquell moment, sense tenir en compte la conveniència futura de tal decisió. Per aquest motiu, en general donen pitjor resultat que altres algorismes d'agrupament.

2.3.4. **k**-mitjanes (**k**-means)

L'algorisme d'agrupament *k*-mitjanes busca una partició de les dades tal que cada punt estigui assignat al grup amb 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* de clústers que volem, ja que per si mateix no és capaç de determinar-lo.

A grans trets l'algorithm 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.

El programa de k -mitjanes està escrit en el codi 4.4. Si bé es tracta d'un algoritme senzill i ràpid, en tenir en compte només 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 particionen l'espai. Així, per a les dades mostrades en la figura 2, la partició en grups aproximada és la mostrada en la figura 5, en la qual es veu 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 dóna en altres punts. A més, el resultat pot variar d'una execució a una altra, ja que depèn dels centroides generats aleatoriament en el primer pas de l'algorisme.

Figura 5. k -mitjanes executat sobre les dades de la figura 2

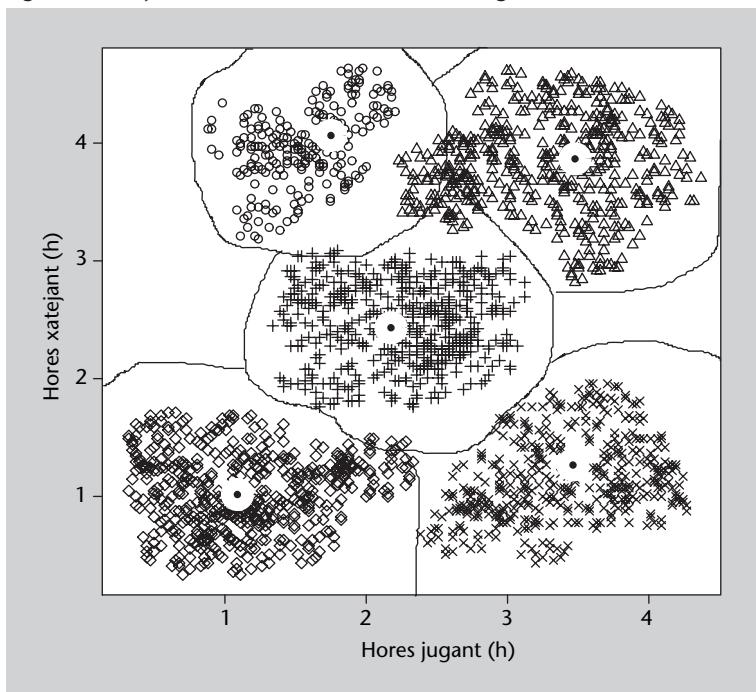


Figura 5

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

2.3.5. *c*-mitjanes difús (*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 diferent grau 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 el qual els graus de pertinença són 1 per al grup principal i 0 per als restants.

L'algorisme de *c*-mitjanes difús és pràcticament idèntic a l'algorisme *k*-mitjanes vist anteriorment; les principals diferències 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 aleatoriament, s'assigna el grau de pertinença de cada punt a cada grup aleatoriament 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 processament d'imatges.

2.3.6. Agrupament espectral (*spectral clustering*)

L'espectre d'una matriu és el conjunt dels seus **valors propis**. La tècnica d'anàlisi de components principals descrita en el subapartat 3.1.2. està estretament relacionada amb l'agrupament espectral, ja que determina els eixos en els quals les dades ofereixen més variabilitat i pren els vectors associats a valors propis més grans; en la tècnica que ens ocupa, per contra, es busquen els valors propis més petits, ja que indiquen escassa variabilitat i, per tant, pertinença a un mateix grup.

Vegeu també

Els valors i vectors propis d'una matriu i l'obtenció amb Python estan explicats en el subapartat 3.1.1.

Si bé el fonament matemàtic detallat queda fora de l'àmbit d'aquests materials, l'algorisme d'agrupament espectral consta dels passos següents:

- 1) Calcular la matriu de distàncies W de les dades que es volen agrupar.
- 2) Calcular la **laplaciana** de la matriu de distàncies mitjançant la fórmula $L_{rw} = D - W$, en què I és la matriu identitat i D és una matriu en la qual els elements de la diagonal són la suma de cada fila de W .
- 3) Calcular els valors i vectors propis de L_{rw} .
- 4) Ordenar els valors propis de més petit a més gran; reordenar les columnes de la matriu de vectors propis segons aquest mateix ordre.
- 5) El nombre de grups k sugerit per l'algorisme és el nombre de valors propis molt petits (menors que 10^{-10} si els grups estan realment separats).
- 6) Aplicar un algorisme de *clustering* convencional (k -mitjanes per exemple) a les k primeres columnes de la matriu de vectors propis, indicant que es volen obtenir k grups.

El codi 2.8 mostra el programa corresponent a la descripció anterior, en la qual faltarà la crida a un dels mètodes d'agrupament vistos anteriorment, com k -mitjanes, per a completar el procés.

Codi 2.8: Agrupament espectral

```

1  from numpy import *
2  from numpy.linalg import eig
3
4
5  # Calcula la distància euclidiana entre dos vectors/tuples
6  def distEuclidiana(v1, v2):
7      return sqrt(sum(pow(x-y, 2) for x,y in zip(v1,v2)))
8
9
10 # Càlcul de la matriu de distàncies d'una llista de punts
11 def matriuDist(punts):
12     # Crea una matriu de  $p \times p$ , i  $p$  és el nombre de punts
13     dist = zeros((len(punts), len(punts)))
14
15     # Calcula les distàncies entre punts
16     for i in range(len(punts)):
17         for j in range(i):
18             d = distEuclidiana(punts[i], punts[j])
19             dist[i,j] = d
20             dist[j,i] = d
21     return dist
22
23
24 # Càlcul de la laplaciana sense normalitzar
25 def laplaciana(W):
26     I = identity(len(W), float)
27     D = diag([sum(Wi) for Wi in W])
28     L = D - W
29     return L
30
31
32 # Generació del model espectral
33 def modelEspectral(punts):
34     # Càlcul dels valors i vectors propis
35     W = matriuDist(punts)
```

Lectura complementària

U. von Luxburg (2007). "A Tutorial on Spectral Clustering". *Statistics and Computing* (vol. 4, núm. 17).

Laplaciana no normalitzada

La laplaciana calculada en l'algorisme d'agrupament espectral es tracta de la laplaciana no normalitzada. Hi ha dues laplacianes més, aplicables en agrupament espectral, la simètrica i la de recorregut aleatori; en molts casos el comportament és equivalent.

```

36     L = laplaciana(W)
37     valp , vecp = eig(L)
38
39     # Ordenació per valor propi (menys a més) dels valors
40     # propis i dels vectors propis (per columnes)
41     ordre = sorted(range(len(valp)), key = valp.__getitem__)
42     valp.sort()
43     vecp2 = zeros((len(vecp), len(vecp)))
44     for col in range(len(ordre)):
45         vecp2[:,col] = vecp[:,orden[col]]
46
47     return valp, vecp2
48
49 # Llegeix un fitxer amb les dades dels jugadors, format
50 # "horesJoc horesXat classe" (la classe s'ignora), i
51 # retorna una llista
52 def llegeixJugadors(nomFitx="Jugadors.txt"):
53     línies = [(l.strip()).split("\t")
54               for l in (open(nomFitx).readlines())
55
56     llista = []
57     for i in range(len(línies)):
58         llista.append((float(línies[i][0]), float(línies[i][1])))
59     return llista
60
61
62 # Conjunt de punts d'exemple en forma de llista
63 punts = [(0.5,4.5), (1,4), (0.5,2), (1,1.5),
64           (2.5,2), (3.5,3.5), (4.5,2.5), (4,1)]

```

L'agrupament espectral és un mètode d'agrupament molt potent, ja que és capaç d'agrupar dades tenint en compte les diferències de densitat, dades convexes (per exemple, un cercle que n'envolta un altre), etc. A més, indica el nombre de grups que s'han de formar.

Habitualment s'aplica un nucli gaussià* en calcular la matriu de distàncies per a millorar-ne el rendiment.

*Vegeu el subapartat 4.5.2.
d'ara endavant.

2.3.7. Recomanadors basats en models

L'aplicació d'algorismes d'agrupament per a la construcció de recomanadors es basa, en general, a utilitzar els grups obtinguts per a decidir què cal recomanar a un usuari, en lloc d'utilitzar totes les dades disponibles. Això agilita enormement el procés de decisió i permet disposar de diverses “vistes” de les dades: grups d'usuaris, de productes, etc. D'aquesta manera, en cada circumstància es pot trobar una recomanació adequada de manera ràpida.

3. Extracció i selecció d'atributs

Suposem que volem analitzar un conjunt de dades per a resoldre un problema determinat. Imaginem, per exemple, una estació meteorològica que cada minut faci una mesura de la temperatura i de la humitat relativa. Al cap d'una hora haurem obtingut un conjunt de dades format per $m = 60$ observacions de $n = 2$ variables, que representarem amb la matriu de mida 60×2 . Si representem la mesura i -èsima de la temperatura amb la variable $a_{i,1}$ i la d'humitat relativa amb $a_{i,2}$, la matriu de dades estarà determinada per

$$\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ \vdots & \vdots \\ a_{60,1} & a_{60,2} \end{pmatrix} \quad (5)$$

Abans de procedir a l'anàlisi de les dades, hem de trobar una manera adequada de representar-les. Al llarg de l'apartat, representarem els conjunts de dades mitjançant una matriu multidimensional A de mida $m \times n$, en la qual les n columnes representen les variables observades i les m files corresponen a les observacions fetes:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & & & \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} \quad (6)$$

Una bona representació de dades resulta essencial perquè, en general, les dades són redundants o contenen informació espúria. Abans de procedir a l'anàlisi, és recomanable obtenir un conjunt reduït de variables que permeti descriure les dades de manera simplificada. En l'exemple anterior, si la temperatura i la humitat mantenen una relació determinada, les dades es podrien simplificar mitjançant una única variable que sigui funció de la temperatura i la humitat.

La simplificació de dades també es coneix com a *reducció de dimensionalitat*, i pretén eliminar aspectes superflus com correlacions entre variables (redundàncies) o fluctuacions estocàstiques (soroll). Aquests aspectes són agrupats en el que s'anomena normalment *soroll*, és a dir, tots aquells factors que no representin informació rellevant del procés que estudiarem.

En el cas anterior, el soroll provindrà de les fluctuacions de temperatura i humitat degudes als fluxos micrometeorològics, o del soroll electrònic introduït pels equips electrònics de mesura.

Per contra, denominarem *senyal* aquells factors que contribueixen de manera rellevant a les dades, de manera que en general es complirà la identitat següent:

$$\text{dades} = \text{senyal} + \text{soroll} \quad (7)$$

Les fonts més habituals de soroll s'associen als processos d'obtenció i transmissió de les dades. Quan les dades són obtingudes per algun tipus de sistema instrumental, s'introduiran certes fluctuacions inherents al sistema físic de mesura o a les etapes d'amplificació i condicionament. Al soroll en els sistemes de mesura se li ha d'afegir el que es produeix durant la transmissió de les dades a través de línies de comunicació, típicament degut a interferències electromagnètiques o a la pèrdua o corrupció de dades en canals analògics o digitals. Finalment, també caldrà considerar qualsevol distorsió de les dades deguda a altres factors com redundàncies, errors o a la incorporació d'informació que no resulti rellevant. Com resulta evident, un dels problemes fonamentals del tractament de dades és separar senyal i soroll. Però aquest no és l'únic aspecte que considerarem en aquest subapartat. De vegades, l'interès no serà l'eliminació del soroll, sinó la identificació dels diferents aspectes que contribueixen a les dades. En aquests casos serà necessari agrupar algunes de les característiques de les dades per a disposar d'una representació d'aquestes en funció de k components elementals més simples que ens faciliten la interpretació

$$\text{dades} = \text{component}_1 + \text{component}_2 + \dots + \text{component}_k. \quad (8)$$

Els components se solen ordenar seguint algun criteri objectiu que en determini la importància, la qual cosa ens permet una interpretació simplificada de les dades mitjançant reconstruccions parcials obtingudes a partir d'alguns dels

components. En general, assumirem que les dades s'obtenen mitjançant la realització de mesures experimentals d'un sistema determinat, de manera que l'estructura de les dades necessàriament reflecteix les característiques d'aquest sistema. Per exemple, en analitzar les dades de l'estació meteorològica anterior, podrem separar les dades en dos factors, un de corresponent als efectes diürns i un altre als nocturns.

L'objectiu, llavors, serà separar les dades en components que donin compte d'alguna de les característiques principals del sistema subjacent. Considerem, per exemple, un conjunt de dades que consisteixi en una seqüència d'imatges per satèl·lit d'una determinada zona geogràfica. Resulta lògic pensar que la presència de diferents característiques geogràfiques a la zona (regions muntanyenques, camps de cultiu, rius, etc.) es reflectiran d'alguna manera en el conjunt de dades per analitzar. Les tècniques que estudiarem en aquest subapartat ens permetran identificar les diferents subzones geogràfiques a partir d'una anàlisi de l'estructura de les dades. El procediment consistirà a definir un subconjunt de característiques de les dades (per exemple, la mida en píxels de zones de color verd en les imatges) per a poder identificar les diferents subzones que es reflecteixen en les dades i, per tant, per a descompondre-les en components diferenciatos.

El nom genèric d'aquest conjunt de tècniques descrites en el subapartat 3.1. és *mètodes de factorització matricial*, perquè descomponen en factors una matriu de dades. Una de les maneres més eficients de separar les dades consisteix a identificar els components que millor en descriuen la variabilitat. L'objectiu serà trobar un sistema de variables que descriguin els eixos en els quals les dades presenten més variabilitat. Aquesta és la idea bàsica de la tècnica de descomposició en components principals (PCA), que descriurem més endavant, en el subapartat 3.1.2. Una altra de les formes consisteix a descompondre les dades en les seves fonts independents, és a dir, en aquells mecanismes de naturalesa independent del sistema subjacent a les dades. Això és el que aprendrem en el subapartat 3.1.3. quan estudiem la tècnica de descomposició en components independents (ICA). La tercera tècnica, que estudiarem en el subapartat 3.1.4., és coneguda com a factorització de matrius no negatives (NMF), i permet una descomposició de les dades de manera que cada component contribueix de manera acumulativa a la matriu global.

En el subapartat 3.3.1. també aprendrem a utilitzar altres tècniques per a la visualització de dades multidimensionals o per a identificar quines variables permeten separar les dades en grups diferents en el subapartat 3.2.1.

3.1. Tècniques de factorització matricial

Les tècniques d'extracció i selecció d'atributs ens permeten descriure les dades en funció de noves variables que posin de manifest alguns dels aspectes de

les dades en els quals estiguem interessats, ometent o atenuant aquells factors que no són rellevants.

Les tècniques que es descriuen en aquest apartat (SVD, PCA, ICA i NMF) són conegudes com a *tècniques de factorització matricial*, ja que totes descomponen una matriu de dades com el producte de matrius més simples. En qualsevol cas, la factorització d'una matriu no és única, i cada tècnica posa de manifest aspectes diferents de la informació continguda en les dades originals.

3.1.1. Descomposició en valors singulars (SVD)

La descomposició en valors singulars és una eina d'àlgebra lineal que té innombrables aplicacions en camps com el càlcul numèric, el tractament de senyals o l'estadística. El motiu pel qual resulta tan útil és que l'SVD permet descompondre una matriu en una forma en què ens resultarà molt senzill calcular-ne els valors i vectors propis (diagonalització de matrius). En sistemes d'intel·ligència artificial, l'SVD permet descompondre una matriu de dades expressant-la com la suma ponderada dels seus vectors propis. Una de les maneres més simples de simplificar un conjunt de dades és descompondre-les utilitzant SVD i reconstruir-les a partir dels vectors propis amb més valor propi. Els vectors propis amb menys valor propi solen correspondre a aspectes de detall de les dades que poden no ser rellevants en una primera exploració. A més, l'SVD també s'utilitza en la descomposició en components principals que descriurem en el subapartat 3.1.2.

En el cas particular d'una matriu quadrada de mida $n \times n$, la *diagonalització* consisteix a trobar el conjunt de valors i vectors propis $\{\lambda_i, v_i\}, i = 1 \dots n$ que compleixin la identitat

$$A \cdot v_i = \lambda_i v_i, \quad (9)$$

per a expressar després la matriu A en la forma

$$A = V^{-1} \cdot \Lambda \cdot V, \quad (10)$$

en què Λ és una matriu diagonal $n \times n$ els components de la qual són els valors propis λ_i i V és una matriu $n \times n$ les n columnes de la qual són els vectors propis v_i (cadascun de mida $1 \times n$).

Diagonalitzar una matriu

Diagonalitzar una matriu consisteix a obtenir-ne els valors i vectors propis. Els vectors propis defineixen un nou sistema de coordenades en el qual la matriu de dades és diagonal.

El codi 3.1 indica com podem obtenir els valors i vectors propis d'una matriu en Python. En la primera línia s'importa la biblioteca *numpy*, que inclou gran part de les funcions que necessitarem en aquest apartat. La línia 3 introduceix una matriu A de mida 3×3 , i en la línia 10 s'obtenen els valors i vectors propis de A utilitzant la rutina *svd* del paquet de routines d'àlgebra lineal *linalg* inclòs en la biblioteca *numpy*. En les instruccions de les línies 20 i 23 es comprova que l'equació 9 es compleix per al primer dels valors propis v_1 (les línies 20 i 23 corresponen, respectivament, als termes esquerre i dret de l'equació 9).

Codi 3.1: Exemple de diagonalització en Python

```

1  >>> from numpy import *
2
3  >>> A = array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
4
5  >>> A
6  array([[1, 2, 3],
7         [4, 5, 6],
8         [7, 8, 9]])
9
10 >>> val,vec = linalg.eig(A)
11
12 >>> val
13 array([ 1.61168440e+01, -1.11684397e+00, -3.21775239e-16])
14
15 >>> vec
16 array([[-0.23197069, -0.78583024,  0.40824829],
17        [-0.52532209, -0.08675134, -0.81649658],
18        [-0.8186735 ,  0.61232756,  0.40824829]])
19
20 >>> dot(A,vec[:,0])
21 array([-3.73863537, -8.46653421, -13.19443305])
22
23 >>> val[0]*vec[:,0]
24 array([-3.73863537, -8.46653421, -13.19443305])

```

Com les matrius de dades no solen ser quadrades, l'SVD generalitza la diagonalització en matrius amb una mida arbitrària $m \times n$. El resultat consisteix en una descomposició factorial de la matriu A de mida $m \times n$ com a producte de tres matrius U , S i V

$$A = U \cdot S \cdot V^T, \quad (11)$$

en què S és una matriu diagonal de mida $m \times n$ amb components no negatius i U, V són matrius unitàries de mida $m \times m$ i $n \times n$, respectivament. Els components diagonals no nuls de la matriu S es coneixen com a *valors singulars*, mentre que les m columnes de la matriu U i les n columnes de la matriu V es denominen *vectors singulars* per l'esquerra i per la dreta, respectivament. El nombre màxim de valors singulars diferents de la matriu A està limitat pel rang màxim de la matriu A , $r = \min\{m,n\}$.

El codi 3.2 descriu com es fa una descomposició SVD en Python utilitzant les eines de la biblioteca *numpy*. El codi comença definint una matriu A de mida

4×2 , i en la línia 8 s'obté la descomposició SVD. L'última instrucció (línia 28) comprova que els termes esquerre i dret de l'equació 11 siguin iguals utilitzant la instrucció *allclose* de Python.

Codi 3.2: Exemple SVD en Python

```

1  >>> from numpy import *
2  >>> A = array([[1,3,5,7],[2,4,6,8]]).T
3  >>> A
4  array([[1,  2],
5       [3,  4],
6       [5,  6],
7       [7,  8]])
8  >>> U,S,Vt = linalg.svd(A, full_matrices=True)
9  >>> U
10 array([[-0.15248323, -0.82264747, -0.39450102, -0.37995913],
11        [-0.34991837, -0.42137529,  0.24279655,  0.80065588],
12        [-0.54735351, -0.0201031 ,  0.69790998, -0.46143436],
13        [-0.74478865,  0.38116908, -0.5462055 ,  0.04073761]])
14 >>> S
15 array([ 14.2690955 ,  0.62682823])
16 >>> Vt
17 array([[ -0.64142303, -0.7671874 ],
18        [ 0.7671874 , -0.64142303]])
19 >>> S1 = zeros((4,2))
20 >>> S1[:2,:2] = diag(S)
21 >>> S1
22 array([[ 14.2690955 ,  0.          ],
23        [ 0.          ,  0.62682823],
24        [ 0.          ,  0.          ],
25        [ 0.          ,  0.          ]])
26 >>> S1.shape
27 (4, 2)
28 >>> allclose(A, dot(U, dot(S1, Vt)))
29 True

```

L'SVD també es pot interpretar com una descomposició en la qual la matriu A s'expressa com la suma de $r = \min\{m,n\}$ matrius, conegudes com a *components SVD*

$$A = U \cdot S \cdot V^T = S(1,1) u_1 \cdot v_1^T + S(2,2) u_2 \cdot v_2^T + \cdots + S(r,r) u_r \cdot v_r^T, \quad (12)$$

en què u_1, u_2, \dots, u_m són els vectors columna de la matriu de vectors singulars per l'esquerra $U = \{u_1, u_2, \dots, u_m\}$, i v_1, v_2, \dots, v_n són els vectors columna de la matriu de vectors singulars per la dreta $V = \{v_1, v_2, \dots, v_n\}$. En aquesta representació, cadascuna de les r matrius $u_j \cdot v_j^T$ té un pes que està determinat pel valor singular corresponent $S(j,j)$. Així, els vectors singulars amb més valor singular seran més representatius de la matriu A , mentre que els vectors singulars amb menys valor singular aportaran molt poc a la descomposició en components SVD de A . Aquest fet és el que permet que SVD sigui utilitzada com a tècnica de representació de dades, ja que resulta una eina útil per a identificar quins vectors singulars són rellevants per a descriure la matriu A i quins es poden considerar com a residuals.

Aplicant aquest procediment a una matriu de dades obtindrem una representació de la informació en funció d'uns pocs vectors singulars, i per tant

disposarem d'una representació de les dades en un espai de dimensionalitat reduïda. Representar les dades de manera fiable en un espai de dimensionalitat reduïda té evidents avantatges per a fer una anàlisi exploratòria de la informació, i pot ser utilitzat com a tècnica de reducció del soroll o de compressió de dades. El codi 3.3 implementa un exemple d'aquesta descomposició en components SVD, i comprova que la matriu A es pot reconstruir com la suma dels seus components (la línia 7 correspon a la part dreta de l'equació 12, el símbol \ és simplement un salt de línia).

Observació

La instrucció `z.reshape(-1,1)` converteix el vector z en un vector columna. Per a convertir un vector z en un vector fila utilitzaríem la instrucció `z.reshape(1,-1)`.

Codi 3.3: Descomposició d'una matriu en els seus components SVD

```

1 #!/usr/bin/env python
2
3 >>> A = array([[1, 2],[3, 4]])
4
5 >>> U,S,Vh = linalg.svd(A)
6
7 >>> S[0]*U[0].reshape(-1,1)*Vh[0] + S[1]*U[1].reshape(-1,1)*Vh[1]
8 array([[ 1.,  2.],
9        [ 3.,  4.]])

```

Una altra propietat interessant de la descomposició SVD de la matriu A és que permet calcular directament els vectors i valors propis de la matriu $A^T \cdot A$. En efecte, tenim que

$$A = U \cdot S \cdot V^T, \quad (13)$$

$$A^T = V \cdot S \cdot U^T, \quad (14)$$

i aplicant la unicitat de U , $U^T \cdot U = 1$, tenim que

$$A^T \cdot A = V \cdot S \cdot U^T \cdot U \cdot S \cdot V^T = V \cdot S^2 \cdot V^T, \quad (15)$$

cosa que demostra que els vectors propis de $A^T \cdot A$ estan determinats (amb la possible excepció del signe) pel vector V^T obtingut en la descomposició SVD de A , i els seus vectors propis per S^2 . Per tant, la descomposició SVD d'una matriu de dades A permet conèixer els valors i vectors propis de la matriu de correlació $A^T \cdot A$. Aquesta propietat es descriu amb un exemple en el codi 3.4, en el qual es comparen els resultats d'aplicar SVD a una matriu A amb els obtinguts de diagonalitzar la matriu $A^T \cdot A$.

Codi 3.4: SVD permet calcular els valors i vectors propis de la matriu $A^T \cdot A$

```

1 >>> from numpy import *
2 >>> A = array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])
3
4 >>> cov_A = dot(A.T,A)
5 >>> cov_A
6 array([[ 66,  78,  90],

```

```

7      [ 78,  93, 108],
8      [ 90, 108, 126]])
9
10 >>> val, vec = linalg.eig(cov_A)
11
12 >>> U, S, Vt = linalg.svd(A)
13
14 >>> val
15 array([ 2.83858587e+02,   1.14141342e+00,  -2.38229079e-15])
16
17 >>> S*S.T
18 array([ 2.83858587e+02,   1.14141342e+00,  1.18098937e-31])
19
20 >>> vec
21 array([[-0.47967118, -0.77669099,  0.40824829],
22        [-0.57236779, -0.07568647, -0.81649658],
23        [-0.66506441,  0.62531805,  0.40824829]])
24
25 >>> Vt.T
26 array([[-0.47967118, -0.77669099,  0.40824829],
27        [-0.57236779, -0.07568647, -0.81649658],
28        [-0.66506441,  0.62531805,  0.40824829]])
```

3.1.2. Anàlisi de components principals (PCA)

Exemple d'aplicació

A manera d'exemple introductori, suposem que disposem d'un conjunt de dades amb informació sobre els jugadors de futbol de primera divisió. Tenint en compte que la lliga espanyola disposa de 20 equips i que cadascun consta de 25 jugadors amb llicència federativa, disposarem d'un total de $m = 500$ observacions. Per a cada jugador disposem de tres variables amb la mitjana de passades, gols i pilotes robades per partit, respectivament ($n = 3$ variables, la matriu de dades tindrà una mida $m = 500 \times 3$). Les dades, representades en el sistema d'eixos tridimensional de les variables, adopten un aspecte similar al d'un núvol de punts en l'espai.

La PCA és una tècnica que permet determinar els eixos principals de la distribució del núvol de punts. Trobar els eixos principals del núvol de punts equival a trobar un nou conjunt de variables en les quals les dades presenten una dispersió màxima, és a dir, aquells eixos en els quals la variabilitat de les dades és més gran. En l'exemple de les dades de futbolistes, aquests eixos corresponen a combinacions de les tres variables en les quals els futbolistes tenen més variabilitat. Podria passar, per exemple, que la majoria de futbolistes facin un nombre mitjà de passades per partit similar, i que les discrepàncies entre jugadors es deguin a una combinació entre el nombre mitjà de pilotes que roben i de gols que marquen. Aquesta combinació de variables descriurà un nou eix de coordenades (el primer component principal), en el qual es produirà més dispersió de les dades en l'espai original. Una projecció de les dades en un subconjunt dels eixos principals permet una simplificació de les dades a partir d'una representació en un espai de dimensionalitat reduïda. Més endavant veurem un exemple genèric en el qual aplicarem la tècnica PCA a un

conjunt de dades tridimensional. La tècnica PCA es basa en la diagonalització de la matriu de covariància de les dades, i les seves característiques principals es descriuen a continuació.

Descripció de la tècnica

Si A representa una matriu de dades de mida $m \times n$ amb m observacions de n variables, la matriu $A^T \cdot A$ és coneguda com a *matriu d'autocorrelació* i expressa les relacions estadístiques existents entre les n variables. Bàsicament, la tècnica PCA consisteix a diagonalitzar la matriu d'autocorrelació $A^T \cdot A$ o la matriu de covariància $(A - \bar{A})^T \cdot (A - \bar{A})$ de les dades A .

Com les matrius d'autocorrelació o autocovariància indiquen correlació estadística entre variables, els seus vectors i valors propis ens permetran obtenir unes noves variables entre les quals la correlació estadística sigui mínima. Els vectors propis donaran lloc als *components principals*, i defineixen uns nous eixos de coordenades orientats en les direccions en les quals les dades presenten més variabilitat. Els vectors propis amb més valor propi correspondran a les direccions en les quals les dades presenten més dispersió, i aquells vectors propis amb un autovalor petit indicaran una direcció en la qual les dades amb prou feines varien el seu valor.

Com hem vist en el subapartat 3.1.1., la descomposició SVD d'una matriu A permet calcular els vectors i valors propis de la matriu $A^T \cdot A$, per la qual cosa l'SVD ens serà de gran utilitat per a descompondre A en els seus components principals.

La relació entre SVD i PCA es posa de manifest en escriure la descomposició SVD de la matriu A en la forma

$$A = R \cdot V^T \quad (16)$$

en què $R = U \cdot S$ és coneguda com a *matriu de resultats* (en anglès, *scores matrix*), i la matriu de vectors singulars per la dreta V^T es coneix com a *matriu de càrregues* (en anglès, *loadings matrix*).

L'equació 16 expressa la descomposició PCA de la matriu A com una factorització en les matrius R i V^T .

Els passos per seguir per a fer una descomposició PCA d'una matriu de dades en Python s'indiquen en l'exemple descrit en el codi 3.5.

Codi 3.5: Exemple de PCA en Python

```

1  from numpy import *
2  import pylab
3  from mpl_toolkits.mplot3d import Axes3D
4  import matplotlib.pyplot as plot
5  set_printoptions(precisio = 3)
6
7  # Dades: distribució normal multivariada en 3d
8  mean = [1,5,10]
9  cov = [[-1,1,2],[-2,3,1],[4,0,3]]
10 d = random.multivariate_normal(mean, cov, 1000)
11
12 # representació gràfica de les dades:
13 fig1 = plot.figure()
14 sp = fig1.gca(projection = '3d')
15 sp.scatter(d[:,0],d[:,1],d[:,2])
16 plot.show()
17
18 # ANALISI PCA:
19 # Pas 1: Calcular la matriu de covariància de les dades (N x N):
20 d1 = d - d.mean(0)
21 matcov = dot(d1.transpose(),d1)
22
23 # Pas 2: Obtenir els valors i vectors propis de la matriu de
24 # covariància:
25 valp1,vecp1 = linalg.eig(mtcov)
26
27 # Pas 3: Decidir quins vectors són els rellevants representant
28 # els valors propis en ordre decreixent (scree plot):
29 ind_creixent = argsort(valp1) # ordre creixent
30 ind_decre = ind_creixent[::-1] # ordre decreixent
31 val_decre = valp1[ind_decre] # valors propis en ordre decreixent
32 vec_decre = vecp1[:,ind_decre] # ordenar també vectors propis
33 pylab.plot(val_decre,'o-')
34 pylab.show()
35
36 # Projectar dades a la nova base definida per
37 # tots els vectors propis (espai PCA 3D)
38 d_PCA = zeros((d.shape[0],d.shape[1]))
39 for i in range(d.shape[0]):
40     for j in range(d.shape[1]):
41         d_PCA[i,j] = dot(d[i,:],vecp1[:,j])
42
43 # recuperar dades originals invertint la projecció (reconstrucció):
44 d_recon = zeros((d.shape[0],d.shape[1]))
45 for i in range(d.shape[0]):
46     for j in range(d.shape[1]):
47         d_recon[i] += d_PCA[i,j]*vecp1[:,j]
48
49 # comprovar que es recuperen les dades originals:
50 allclose(d,d_recon)
51
52 # Projectar dades a la nova base definida pels dos
53 # vectors propis amb més valor propi (espai PCA 2D)
54 d_PCA2 = zeros((d.shape[0],2))
55 for i in range(d.shape[0]):
56     for j in range(2):
57         d_PCA2[i,j] = dot(d[i,:],vec_decre[:,j])
58
59 # reconstruir dades invertint la projecció PCA 2D:
60 d_recon2 = zeros((d.shape[0],d.shape[1]))
61 for i in range(d.shape[0]):
62     for j in range(2):
63         d_recon2[i] += d_PCA2[i,j]*vec_decre[:,j]
64
65 # representació gràfica de les dades:
66 fig2 = plot.figure()
67 sp2 = fig2.gca(projection = '3d')
68 sp2.scatter(d_recon2[:,0],d_recon2[:,1],d_recon2[:,2],c='r',marker='x')
69 plot.show()

```

Nou sistema de coordenades

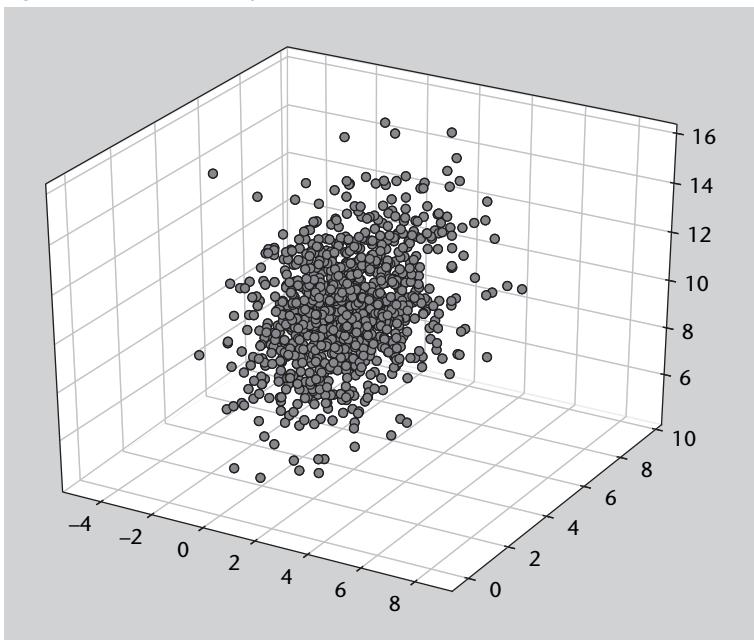
La PCA permet caracteritzar un conjunt de dades mitjançant un nou sistema de coordenades en les direccions en les quals les dades presenten més variabilitat.

En la línia 10 es genera una matriu 1000×3 de dades distribuïdes aleatòriament segons una densitat de probabilitat normal multivariada amb mitjana $(1,5,10)$ i matriu de covariància

$$\begin{pmatrix} -1 & 1 & 2 \\ -2 & 3 & 1 \\ 4 & 0 & 3 \end{pmatrix}. \quad (17)$$

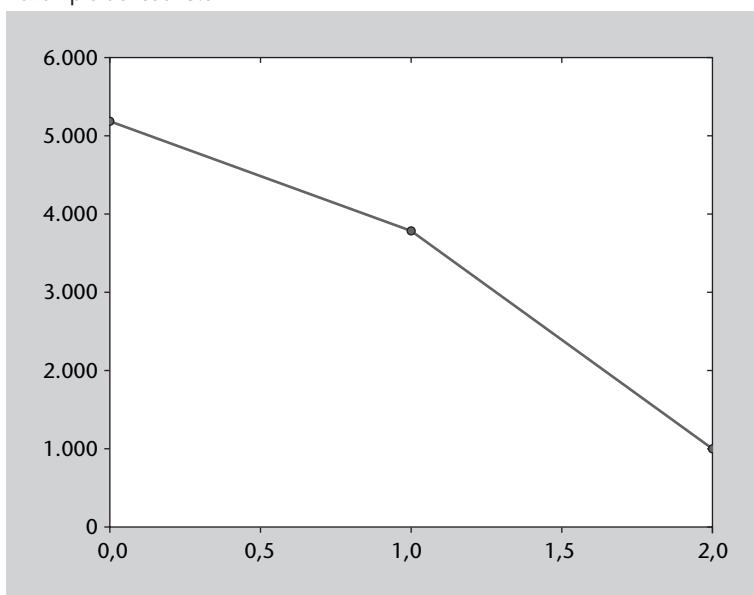
Aquesta matriu de dades és similar a la descrita en l'exemple de les dades de futbolistes al qual fèiem referència en la introducció d'aquest subapartat. Les dades es representen en la figura 6 mitjançant una gràfica tridimensional, en la qual s'aprecia la distribució de dades en forma de núvol de punts.

Figura 6. Dades de l'exemple PCA descrit en el codi 3.5



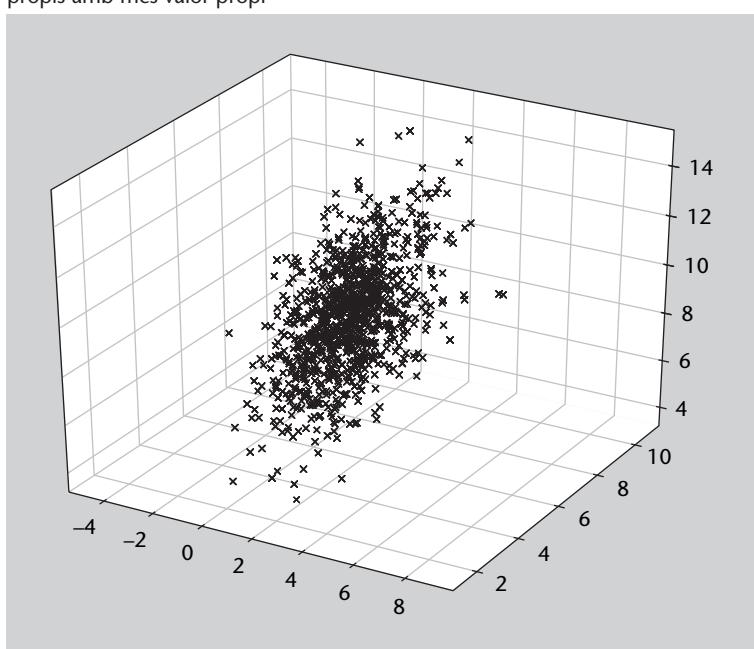
El primer pas per a fer PCA és obtenir la matriu d'autocovariància de les dades (línia 21), que s'obté després de sostreure la mitjana a cada columna de la matriu d (línia 20). El segon pas és obtenir els valors i vectors propis de la matriu d'autocovariància (línia 24). Per a decidir quins vectors propis són més rellevants, l'habitual és representar gràficament els valors propis en ordre decreixent, en el que es coneix com un *scree plot* (diagrama de pedregar, per la manera abrupta com decreixen els autovalors, que recorda al vessant d'una muntanya). Els valors i vectors propis són ordenats seguint un ordre decreixent del valor propi (línies 28-31). Aquest diagrama apareix representat en la figura 7, en la qual s'observa que un dels autovalors és significativament més petit que els altres dos. Cadascun dels vectors propis és un vector que apunta en tres direccions ortogonals en les quals les dades presenten més variabilitat (espai PCA).

Figura 7. Scree plot dels valors propis de la matriu d'autocovariància de l'exemple del codi 3.5



En les línies 37 a 40 es projecten les dades originals a l'espai PCA fent el producte escalar entre les dades i els vectors propis (línia 40). Les dades originals es poden recuperar invertint la projecció (línia 43), la qual cosa es comprova en la línia 49 del codi. A continuació, es fa una projecció de les dades a un espai PCA que només té en compte els dos vectors propis amb més valor propi associat (línies 53-56). En considerar els dos components més rellevants de la reconstrucció de les dades a partir de l'espai PCA de dimensionalitat reduïda s'assembla a les dades originals (figura 8).

Figura 8. Reconstrucció de les dades de l'exemple 3.5 utilitzant els dos vectors propis amb més valor propi



Exemple: PCA per a imatges

Una de les aplicacions més interessants de la PCA és l'anàlisi d'imatges. En aquest cas, la PCA s'aplica a una seqüència d'imatges amb l'objectiu d'identificar les característiques principals que hi apareixen. Les aplicacions són nombroses, i inclouen aspectes com el reconeixement de cares, l'anàlisi d'imatges mèdiques o el processament d'informació geogràfica obtinguda mitjançant tècniques de teledetecció per satèl·lit.

Matemàticament, una imatge no és més que una matriu de $n \times m$ valors enteros. Cadascun dels components de la matriu representa un píxel de la imatge, i adopta valors enteros entre 0 i $2^k - 1$, en què k és el nombre de bits utilitzats per a representar cada píxel. En cas que s'utilitzin 8 bits per píxel ($k = 8$), per exemple, els components de la matriu prendran valors enteros en el rang $[0 - 255]$. Per tant, una seqüència d'imatges no és més que un conjunt de NIM matrius, cadascuna de mida $n \times m$. Per a poder treballar amb PCA, les dades s'organitzen de manera que cada imatge sigui emmagatzemada en un únic vector fila que contingui les seves NPIX = $n \cdot m$ pixels. D'aquesta manera, la seqüència d'imatges es descriu matemàticament com una matriu de mida $NIM \times NPIX$ que adopta la forma

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,NPIX} \\ a_{2,1} & a_{2,2} & \dots & a_{2,NPIX} \\ a_{3,1} & a_{3,2} & \dots & a_{3,NPIX} \\ \vdots & & & \\ a_{NIM,1} & a_{NIM,2} & \dots & a_{NIM,NPIX} \end{pmatrix}, \quad (18)$$

en què cada fila correspon a una imatge de la seqüència i cada columna representa un dels pixels de la imatge.

El problema principal amb el qual ens trobem en aplicar PCA a una seqüència d'imatges és que normalment el nombre d'imatges NIM és considerablement més petit que el nombre de pixels NPIX de cada imatge. El problema és que per a calcular PCA caldria diagonalitzar la matriu de covariància de mida $NPIX \times NPIX$, la qual cosa resulta tremedament costós fins i tot per a imatges de mida reduïda (amb imatges de 128×128 pixels hauríem de diagonalitzar una matriu de 16384×16384). Per a solucionar aquesta situació, s'utilitza una interessant propietat algebraica segons la qual les matrius $A^T \cdot A$ (de mida $NPIX \times NPIX$) i $A \cdot A^T$ ($NIM \times NIM$) comparteixen els valors propis no nuls. En efecte, les condicions de diagonalització de totes dues matrius estan determinades per

Imatges NIM

Una seqüència de NIM imatges de NPIX pixels es pot representar com una matriu de dades amb NPIX variables i NIM observacions.

$$A^T \cdot A \cdot u_i = \lambda_i u_i \quad (19)$$

$$A \cdot A^T \cdot w_i = \lambda_i w_i. \quad (20)$$

Multiplicant la segona equació per A^T per l'esquerra tenim

$$A^T \cdot A \cdot A^T \cdot w_i = A^T \lambda_i w_i, \quad (21)$$

i identificant 21 amb l'equació de 19 tenim que els valors propis de $A^T \cdot A$ (u_i) i els de $A \cdot A^T$ (w_i) estan relacionats de la manera

$$A^T \cdot w_i = u_i \quad (22)$$

per la qual cosa podem obtenir els vectors propis de $A^T \cdot A$ diagonalitzant la matriu $A \cdot A^T$ que, en el cas d'imatges, té una mida més petita. Aquesta estratègia és la que s'implementa en el codi 3.6, en el qual s'aplica PCA a una seqüència d'imatges. El codi suposa que les imatges tenen extensió *.jpg i que es troben en el mateix directori en el qual s'executi el programa. Les línies 13-23 construeixen una llista Python amb els noms dels fitxers amb extensió *.jpg trobats en el directori. Després de determinar el nombre d'imatges NIM i la mida NPIX, es construeix una matriu de dades de mida $NIM \times NPIX$ amb les NIM imatges desplegades en forma de vectors fila de longitud NPIX (línia 33). A continuació es procedeix a centrar les dades (línies 41-43), s'obté la matriu $M = A \cdot A^T$ (línia 45) i es calculen els vectors i valors propis de la matriu M (línia 46). Els vectors propis de $A^T \cdot A$ s'obtenen utilitzant l'equació 22 (línia 47). Les línies 49 ordenen els vectors propis en ordre decreixent dels seus valors propis, i la següent determina els valors propis de la matriu de dades A .

Codi 3.6: Exemple d'aplicació de PCA per a l'anàlisi d'imatges

```

1   import os
2   from PIL import Image
3   import numpy
4   import pylab
5   from pca_im import *
6   import matplotlib.pyplot as plt
7
8 ######
9 # CONSTRUIR Matriu DE DADES
10 #####
11
12 # Obtenir directori actual en què són les imatges
13 path= os.getcwd()
14 dirList=os.listdir(path)
15
16 # construir una llista amb els fitxers amb extensió '.jpg'
17 imlist = []
18 i = 0
19 for fname in dirList:
20     filename, filext = os.path.splitext(fname)
21     if filext == '.jpg':

```

```

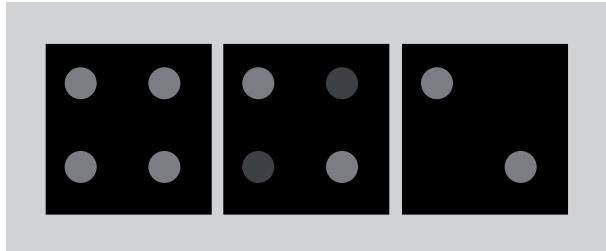
22         imlist.append(fname)
23         i = i +1
24
25 # Obrir la primera imatge i obtenir-ne la grandària en píxels
26 im = numpy.array(Image.open(imlist[0]))
27 m,n = im.shape[0:2]
28
29 NPIX = n*m    # Nombre de píxels de cada imatge:
30 NIM = len(imlist) # Nombre dd'imatges que s'han d'analitzar:
31
32 # Crear una matriu de grandària NIM x NPIX:
33 A = numpy.array([numpy.array(Image.open(imlist[i])).flatten()
34                 for i in range(NIM)], 'f')
35
36 #####
37 # ANÀLISI PCA
38 #####
39
40 # Centratament de les dades restant la mitjana:
41 im_mitjana = A.mean(axis=0)
42 for i in range(NIM):
43     A[i] -= im_mitjana
44
45 M = dot(A,A.T) # matriu AA' (NIM x NIM)
46 lam,vec = linalg.eigh(M) # obtenir els NIM vectors i valors propis de M
47     = AA'
48 aux = dot(A.T,vec).T # aplicar w = A'v per a obtenir els vectors propis
49     de A'A
50 V = aux[::-1] # ordenar vectors propis
51 S = sqrt(lam)[::-1] # ordenar valors propis de A
52
53 # Recompondre la imatge mitjana:
54 im_mitjana = im_mitjana.reshape(m,n)
55
56 # Representar els autovalors PCA
57 pylab.plot(S[0:10], 'o-')
58
59 # Obtenir els dos primers modes PCA i representar-los:
60 mode1 = V[0].reshape(m,n)
61 mode2 = V[1].reshape(m,n)
62
63 # Representar gràficament:
64 fig1 = pylab.figure()
65 fig1.suptitle('Imatge_mitjana')
66 pylab.gray()
67 pylab.imshow(im_mitjana)
68 ## 
69 fig2 = pylab.figure()
70 fig2.suptitle('Primer_mode_PCA')
71 pylab.gray()
72 pylab.imshow(mode1)
73 ## 
74 fig3 = pylab.figure()
75 fig3.suptitle('Segon_mode_PCA')
76 pylab.gray()
77 pylab.imshow(mode2)
78
79 pylab.show()

```

L'anàlisi d'imatges amb PCA treballa amb una matriu de dades $NIM \times NPIX$, i es pot interpretar com la descomposició en components principals de $NPIX$ variables de les quals tenim NIM observacions. Els vectors propis indiquen les combinacions lineals de píxels que presenten més variabilitat en la seqüència. Així, un píxel que adopti un valor constant durant la seqüència d'imatges no serà inclòs en cap dels components amb més valor propi. En canvi, aquells píxels o grups de píxels que presentin més variabilitat al llarg de la seqüència

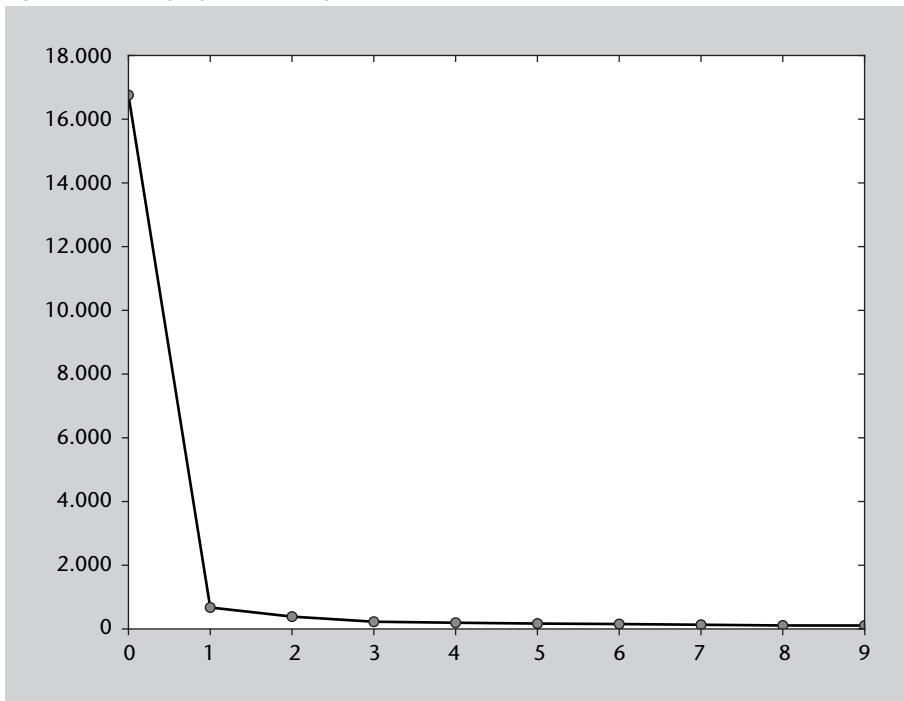
donaran lloc als components principals de la descomposició. Això és el que es pot observar quan apliquem el codi 3.6 a una seqüència de 125 imatges com les que es representen en la figura 9.

Figura 9. Imatges 1, 4 i 7 de la seqüència de l'exemple 3.6



En aquest exemple, els píxels dels cercles superior esquerre i inferior dret adopten un valor constant durant la seqüència, mentre que els cercles superior dret i inferior esquerre canvien el seu valor seguint un senyal sinusoïdal, i completen quatre períodes al llarg de la seqüència. La figura 10 representa els deu primers valors propis que s'obtenen en aplicar el codi PCA 3.6 a la seqüència d'imatges de l'exemple.

Figura 10. Valors propis de l'exemple 3.6



El primer valor propi és significativament més gran que la resta, la qual cosa ens indica que la major part de la variabilitat de la seqüència es pot resumir en un únic component. La reconstrucció que s'obté a partir d'aquest primera component es representa en la figura 11, i conté únicament els cercles superior dret i inferior esquerre, que són els que canvien al llarg de la seqüència. El segon mode PCA, amb un valor propi molt més petit, es representa en la figura 12 i correspon a valors residuals en l'entorn del primer component PCA.

Ni el fons de la imatge ni els cercles superior esquerre i inferior dret apareixen representats en els cinc primers modes de l'anàlisi PCA a causa que adopten un valor constant durant la seqüència.

Figura 11. Primer mode PCA de l'exemple 3.6

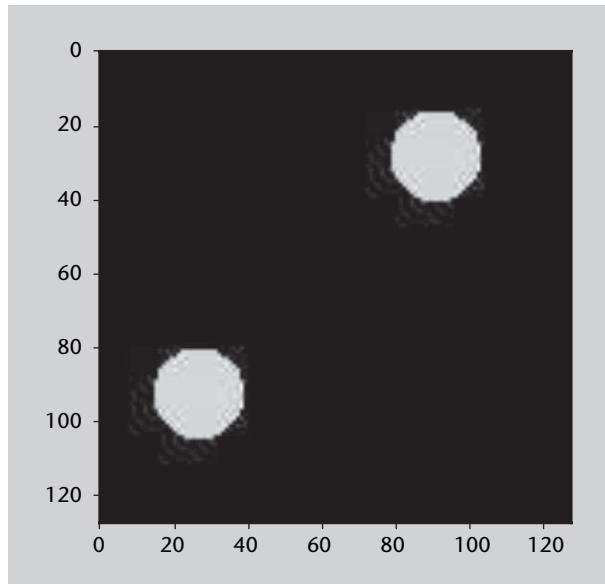
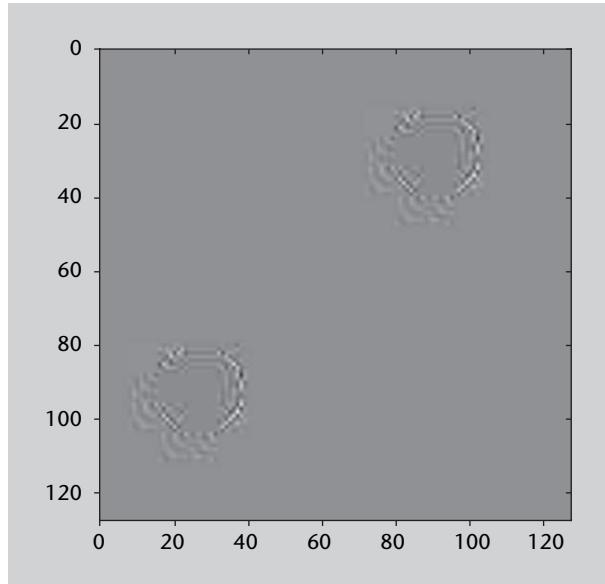


Figura 12. Segon mode PCA de l'exemple 3.6



Anàlisi PCA d'una seqüència de cares: Eigenfaces (autocares)

Sens dubte, l'exemple paradigmàtic de PCA per a imatges és l'anàlisi d'una seqüència de cares. En aquest exemple partim d'una seqüència de 25 imatges de cares obtingudes de la base de dades AR Face Database de la Universitat d'Ohio. Les 25 cares es representen en la figura 13 i corresponen a un conjunt de persones amb diferents expressions facials (somrient, seriós, cridant, etc.).

Lectura recomanada

A. M. Martinez; R. Benavente(juny, 1998). *The AR Face Database*. CVC Technical Report (núm. 24).

Figura 13. Seqüència d'imatges original (Exemple 3.7)



Igual que en l'exemple anterior, el conjunt de dades de partida consta de NIM files i $NPIX$ columnes (matriu A , codi 3.7). El codi PCA és similar al de l'exemple anterior, però en aquest cas la descomposició PCA es fa mitjançant la descomposició SVD de la matriu de dades A . En el codi 3.7 de l'exemple, les línies 13-38 es dediquen a obrir les imatges contingudes en un directori i a construir la matriu de dades A . L'anàlisi PCA es fa a partir de la línia 44, començant per centrar les dades i seguint amb la descomposició SVD de la matriu A (línia 55). Tal com es va explicar en el subapartat 3.1.1. (equació 15 i el codi 3.4), la descomposició SVD de A permet trobar els valors i vectors propis de les matrius $A^T A$ i AA^T . Les línies 58-68 simplement comproven que totes dues matrius es recuperen a partir dels vectors singulars per la dreta (V) i per l'esquerra (U), respectivament. A partir de la línia 70 es procedeix a la reconstrucció PCA de la matriu de dades A . Quan s'utilitzen tots els vectors propis (*eigenfaces*), la reconstrucció recupera la matriu de dades original A (línia 75).

Una reconstrucció parcial a partir d'un conjunt reduït d'*eigenfaces* permet fer una descripció simplificada de la matriu de dades A i, per tant, de la seqüència d'imatges original. En les línies 78-81 es reconstrueix la matriu de dades a partir d'un nombre de components igual a 24, 10, 2 o 1 únic mode PCA (1 única *eigenface*). Per a observar com de fiables són cadascuna de les reconstruccions, les figures representen les imatges reconstruïdes. És important notar que com la descomposició SVD s'ha aplicat a la matriu de dades centrada (havent extret la mitjana), a les reconstruccions finals cal afegir la imatge mitjana per a recuperar la imatge correcta (línia 90 i següents de la secció de figures del codi 3.7).

Codi 3.7: Exemple *eigenfaces*: descomposició PCA d'una seqüència d'imatges amb cares

```

1 import os
2 from PIL import Image
3 from PIL import ImageOps
4 import numpy
5 import pylab
6 from pca_im import *
7 import matplotlib.pyplot as plt
8 #####
9 # CONSTRUIR MATRIU DE DADES
10 #####
11 # Obtenir directori actual en el qual són les imatges
12 path= os.getcwd()
13 dirList=os.listdir(path)
14
15 # construir una llista amb els fitxers amb extensió '.jpg'
16 imlist = []
17 i = 0
18 for fname in dirList:
19     filename, filext = os.path.splitext(fname)
20     if filext == '.jpg':
21         imlist.append(fname)
22         i = i +1
23
24 # Obrir la primera imatge i obtenir-ne la mida en píxels
25 z = Image.open(imlist[0])
26 # Convertir a escala de grisos (PIL, ImageOps)
27 ImageOps.grayscale(z)
28 im = numpy.array(z)
29 m,n = im.shape[0:2]
30
31 NPIX = n*m    # Nombre de píxels de cada imatge:
32 NIM = len(imlist) # Nombre d'imatges a analitzar:
33
34 # Crear una matriu de mida NIM x NPIX:
35 A = numpy.array([numpy.array(ImageOps.grayscale(Image.open(imlist[i]))) for i in range(NIM)], 'f')
36
37 #####
38 # ANÀLISI PCA
39 #####
40
41 # Centrat de les dades restant la mitjana:
42 im_mitjana = A.mean(axis=0)
43 for i in range(NIM):
44     A[i] -= im_mitjana
45
46 im_mitjana = im_mitjana.reshape(m,n)
47
48 AAT = dot(A,A.T) # matriu AA' (NIM x NIM)
49 ATA = dot(A.T,A) # matriu A'A (NPIX x NPIX)
50
51 # Calcular PCA a partir de la descomposició SVD de A:
52 U,S,Vt = linalg.svd(A)
53 SS = S*S.T
54
55 # reconstrucció matriu ATA:
56 S2 = zeros((NPIX,NPIX))
57 for i in range(size(SS)):
58     S2[i,i] = SS[i]
59 resultATA = dot(Vt.T,dot(S2,Vt))
60
61 # reconstrucció matriu AAT:
62 S3 = zeros((NIM,NIM))
63 for i in range(size(SS)):
64     S3[i,i] = SS[i]
65 resultAAT = dot(U,dot(S3,U.T))
66
67
68

```

```

69
70 # reconstrucció matriu A:
71 S1 = zeros(A.shape)
72 for i in range(size(S)):
73     S1[i,i] = S[i]
74
75 resultA = dot(U,dot(S1,Vt)) # recuperar matriu original
76
77 # reconstruccions parcials a partir d'uns quants modes PCA:
78 reconA24 = dot(U[:, :24],dot(S1[:, :24],Vt[:, :24]))
79 reconA10 = dot(U[:, :10],dot(S1[:, :10],Vt[:, :10]))
80 reconA2 = dot(U[:, :2],dot(S1[:, :2],Vt[:, :2]))
81 reconA1 = dot(U[:, :1],dot(S1[:, :1],Vt[:, :1]))
82
83 # Figures
84
85 # reconstrucció de la primera imatge de la seqüència:
86 fig1 = pylab.figure()
87 pylab.gray()
88 pylab.axis('off')
89 fig1.suptitle('Imatge_n1_original')
90 pylab.imshow(A[0].reshape(m,n)+im_mitjana)
91
92 fig2 = pylab.figure()
93 pylab.gray()
94 pylab.axis('off')
95 fig2.suptitle('Imatge_n1_reconstrucció_PCA_24_modes')
96 pylab.imshow(reconA24[0].reshape(m,n)+im_mitjana)
97
98 fig3 = pylab.figure()
99 pylab.gray()
100 pylab.axis('off')
101 fig3.suptitle('Imatge_n1_reconstrucció_PCA_10_modes')
102 pylab.imshow(reconA10[0].reshape(m,n)+im_mitjana)
103
104 fig4 = pylab.figure()
105 pylab.gray()
106 pylab.axis('off')
107 fig4.suptitle('Imatge_n1_reconstrucció_PCA_2_modes')
108 pylab.imshow(reconA2[0].reshape(m,n)+im_mitjana)
109
110 fig5 = pylab.figure()
111 pylab.gray()
112 for i in range(NIM):
113     fig5.suptitle('Conjunt_d'imatges_original')
114     pylab.subplot(5,5,i)
115     pylab.imshow(A[i].reshape(m,n)+im_mitjana)
116     pylab.axis('off')
117
118 # reconstrucció_de_totes_les_imatges_de_la_seqüència
119
120 fig6_=pylab.figure()
121 pylab.gray()
122 fig6_.suptitle('reconstrucció PCA 10 modes')
123 for _i_in_range(NIM):
124     pylab.subplot(5,5,_i)
125     pylab.imshow(reconA10[_i].reshape(m,n)+im_mitjana)
126     pylab.axis('off')
127
128 fig7_=pylab.figure()
129 pylab.gray()
130 pylab.axis('off')
131 fig7_.suptitle('reconstrucció PCA 2 modes')
132 for _i_in_range(NIM):
133     pylab.subplot(5,5,_i)
134     pylab.imshow(reconA2[_i].reshape(m,n)+im_mitjana)
135     pylab.axis('off')
136
137 pylab.show()

```

En la figura 14, per exemple, es mostra la primera imatge de la seqüència (a dalt esquerra), i també les reconstruccions parcials a partir de 24 modes (a dalt dreta), 10 modes (a baix esquerra) o 2 modes (a baix dreta). Resulta evident que a mesura que es va reduint el nombre de modes, la reconstrucció de la cara és més diferent a la imatge original. De fet, una reconstrucció amb 24 modes és gairebé perfecta, la qual cosa resulta evident, ja que el nombre d'imatges $NIM = 25$ i, com hem explicat anteriorment, només hi ha 25 vectors propis amb valors propis no nuls.

Figura 14. Reconstrucció PCA de la primera cara de l'exemple 3.7



Figura 14

A dalt esquerra: imatge original. A dalt dreta: reconstrucció PCA a partir de 24 modes. A baix esquerra: reconstrucció PCA 10 modes. A baix dreta: reconstrucció PCA 2 modes.

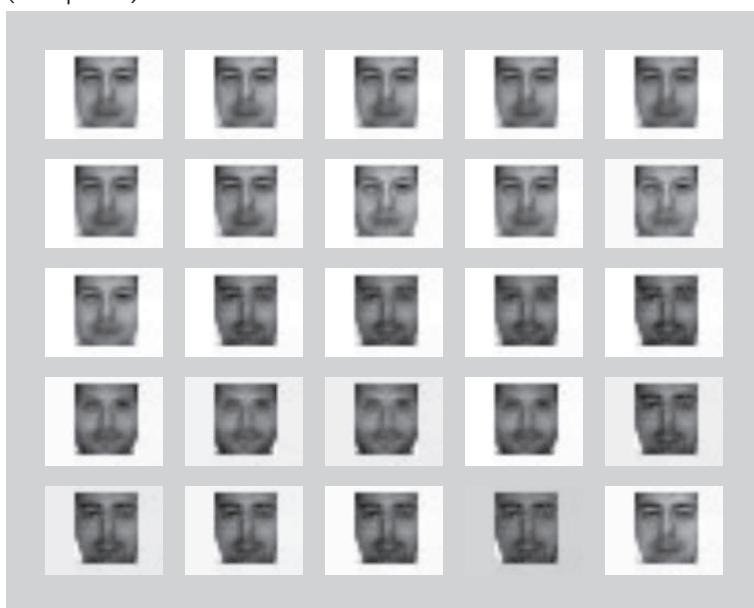
Les figures 15 i 16 mostren la reconstrucció del conjunt de dades complet a partir de 10 i 2 modes, respectivament. Comparant amb la seqüència d'imatges original (figura 13), resulta evident que la reconstrucció amb 10 modes resulta suficient per a capturar les diferents expressions facials de la seqüència original. Una reconstrucció a partir de 2 modes, en canvi, presenta deficiències evidents i només capture els trets generals de les cares, sense permetre una distinció clara de les expressions facials o de les característiques de cada una de les persones. En aquest context, la tècnica PCA s'utilitza tant per a reduir la mida de les imatges*, com per a disposar d'un conjunt reduït de característiques que permeten discriminar entre diferents persones o diferents expressions facials. El nombre d'*eigenfaces* escollides depàndrà dels objectius concrets de cada aplicació.

*Reduir el nombre de components equival a reduir el nombre de components no nuls de la matriu de dades i, per tant, de la mida necessària per a emmagatzemar les dades.

Figura 15. Reconstrucció de la seqüència d'imatges a partir de 10 modes PCA
(Exemple 3.7)



Figura 16. Reconstrucció de la seqüència d'imatges a partir de 2 modes PCA
(Exemple 3.7)



3.1.3. Anàlisi de components independents (ICA)

Exemple d'aplicació

Com a exemple d'aplicació d'aquesta tècnica, considerem una festa en la qual diverses persones parlen de manera simultània. En aquesta festa cada persona percep un senyal acústic en el qual es barregen les veus de la resta dels assistents. Perquè dues persones puguin mantenir una conversa, és necessari que

cadascuna identifiqui les paraules emeses pel seu interlocutor. Com cada persona rep una barreja de tots els assistents, el cervell ha de fer algun tipus de descomposició que li permeti identificar el senyal emès per la persona amb la qual està parlant en aquest moment. El cervell fa aquesta separació, a més de per característiques acústiques (intensitat de la veu, to) o visuals (moviment dels llavis), pel fet que cada persona emet un senyal que, *estadísticament*, és *independent* de la de la resta dels assistents. En efecte, encara que les converses comparteixin una temàtica comuna, el fet que cada assistent és un sistema físic independent garanteix que els seus senyals siguin estadísticament independents de la resta.

L'anàlisi de components independents (ICA) és una tècnica estadística que permet identificar fonts estadísticament independents a partir d'un conjunt de senyals barrejats. En altres paraules, ICA és una tècnica de gran utilitat que permet descompondre un determinat senyal en les fonts *independents* que el componen. ICA també es coneix com a *separació cega de fonts**. En termes generals, ICA és una tècnica que utilitza la *independència estadística* per a identificar les diferents fonts que contribueixen a una determinada barreja de senyals.

*En anglès, *blind source separation* (BSS).

Convé fer ressaltar aquí la importància del terme *independent*. Les tècniques descrites anteriorment (SVD, PCA) descomponen en factors que no eren necessàriament independents entre si des d'un punt de vista estadístic.

Descripció de la tècnica

Com hem comentat anteriorment, ICA és un mètode que resulta útil quan les fonts per identificar provenen de sistemes diferents que emeten senyals independents. Aquesta és la primera hipòtesi d'aplicació d'aquest mètode. La segona condició necessària és disposar d'un nombre de senyals barrejats igual al nombre de components independents que volem identificar.

Vegeu també

En el subapartat següent explicarem amb més detall què s'entén per *estadísticament independent* des d'un punt de vista matemàtic, la qual cosa ens permetrà formular i comprendre les bases de la tècnica ICA.

El subapartat següent està dedicat a aclarir alguns aspectes bàsics sobre l'estadística de senyals, i en particular sobre els conceptes d'independència i correlació estadística. Tornarem a l'exemple d'aplicació de la tècnica ICA més endavant en aquest subapartat. Aquells estudiants que disposin de coneixements sobre estadística i variables aleatòries es poden dirigir directament a la descripció del codi ICA en Python i a l'exemple d'aplicació descrit en el subapartat 3.1.3.

Independència i correlació estadística

En el llenguatge col·loquial, dos senyals que no estan correlacionats se solen anomenar independents. Des del punt de vista matemàtic, l'affirmació anterior no és certa: independència i correlació són propietats estadístiques diferents.

Independència i falta de correlació

Dues variables estadísticament independents no presenten correlació, però l'absència de correlació no implica necessàriament independència estadística. Independència i falta de correlació són, per tant, conceptes matemàticament diferents.

En concret, la independència estadística és una condició més restrictiva que l'absència de correlació, ja que dos senyals independents necessàriament estaran descorrelacionats però dos senyals descorrelacionats no necessàriament han de ser independents.

Per a formular aquest concepte de manera precisa cal treballar amb les *densitats de probabilitat* de cadascun dels senyals. Suposem, doncs, que coneixem els n valors $\{x_1, x_2, \dots, x_n\}$ que un senyal $x(t)$ pren en un conjunt discret de temps $\{t_1, t_2, \dots, t_n\}$. L'*histograma* de valors de $x(t)$ es construeix a partir del nombre d'instants t_i en què el senyal pren valors en un determinat interval. Si els valors màxim i mínim del senyal estan determinats per x_{min} i x_{max} , respectivament, dividirem el rang de valors que pren el senyal $x(t)$ en m intervals de mida $\Delta x = (x_{max} - x_{min})/m$. L'interval k -èsim està determinat per

$$b_k = [x_{min} + (k-1)\Delta x, x_{min} + k\Delta x], k = 1 \dots m, \quad (23)$$

de manera que el primer interval és $b_1 = [x_{min}, x_{min} + \Delta x]$ i l'últim $b_m = [x_{max} - \Delta x, x_{max}]$. Si el senyal $x(t)$ pren un total de h_k valors en l'interval b_k , l'*histograma* de $x(t)$ estarà determinat per $\{h_1, h_2, \dots, h_m\}$. El codi 3.8 indica com podem calcular histogrames i funció de densitat de probabilitat en Python. En aquest exemple es parteix d'un conjunt de dades gaussianes amb mitjana $\mu = 200$ i desviació típica $\sigma = 30$.

Codi 3.8: Càlcul d'*histograma* i funció densitat de probabilitat en Python

```

1 import numpy as np
2 import matplotlib.mlab as mlab
3 import pylab
4
5 # dades gaussianes mitjana mu desviació estàndard sigma:
6 mu, sigma = 200, 30
7 y_dades = mu + sigma*np.random.randn(10000)
8
9 # histograma recompte de valors en cada bin:
10 nbins = 50 #nombre d'intervals
11
12 fig1 = pylab.figure()
13 n, bins, patches = pylab.hist(y_dades, nbins, normed=0, facecolor='green',
14                               alpha=0.75)
14 pylab.title(r'$\mathbf{Histograma\,dades\,gaussianes}\,\mu=200,\,\sigma=30$')
15 pylab.ylabel('Comptatge')
16 pylab.xlabel('Intervals_(bin)')
17 pylab.show()
18
19 # histograma normalitzat de les dades (densitat de probabilitat)
20 fig2 = pylab.figure()
21 p, bins, patches = pylab.hist(y_dades, nbins, normed=1, facecolor='blue',
22                               alpha=0.75)
22
23 # Ajust de les dades amb una funció de densitat
24 # de probabilitat gaussiana:
```

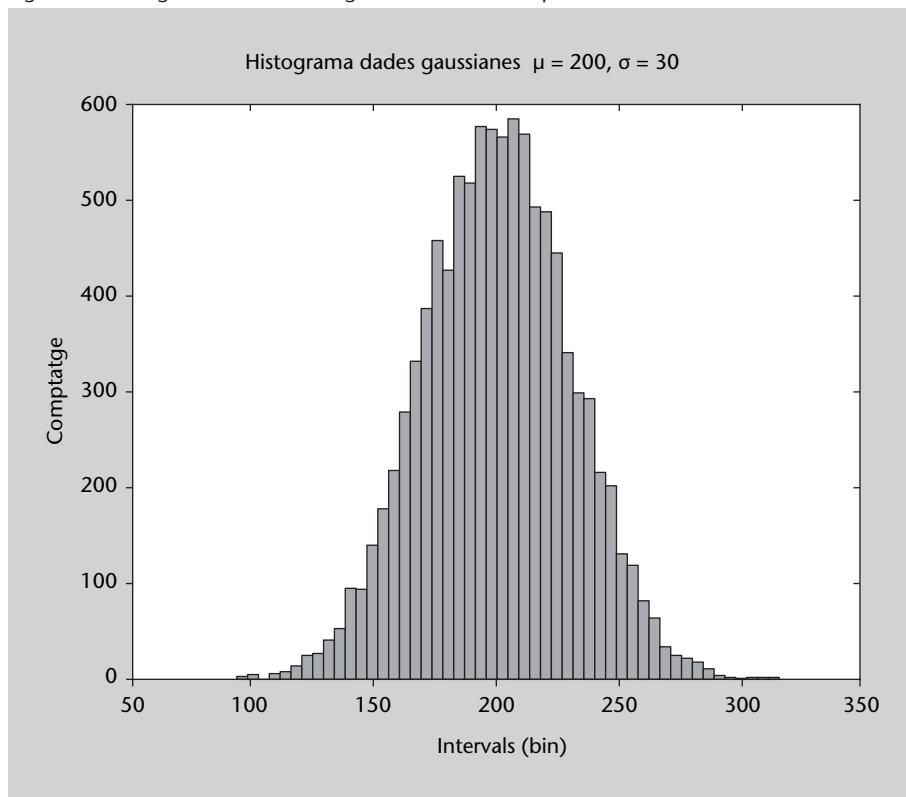
```

25 y_pdf = mlab.normpdf(bins, mu, sigma)
26 l = pylab.plot(bins, y_pdf, 'r--', linewidth=1)
27 pylab.ylabel('Probabilitat')
28 pylab.xlabel('Intervals_(bin)')
29 pylab.title(r'$\mathrm{Densitat\_de\_probabilitat} \mu=200, \sigma=30$')
30 pylab.grid(True)
31 pylab.show()
32
33 # Comprovar que la integral de la funció densitat de probabilitat
34 # és igual a 1:
35
36 Integral = sum(p*np.diff(bins))
37 print Integral

```

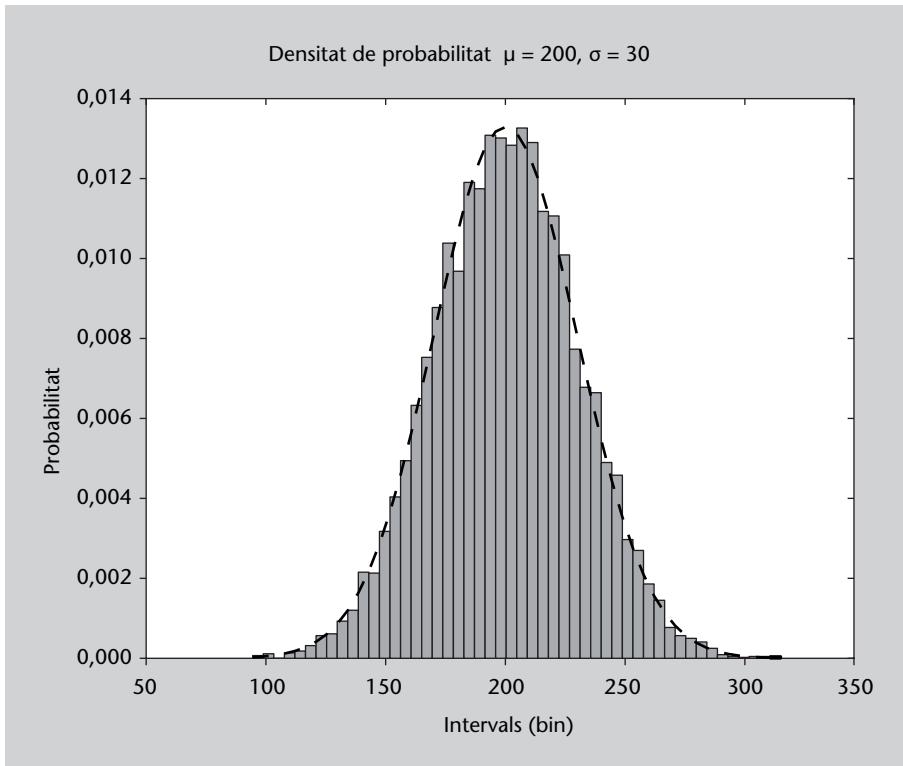
La figura 17 representa el nombre de dades que prenen valors dins de cada interval (comptatge).

Figura 17. Histograma de les dades gaussianes de l'exemple 3.8



En la figura 18 l'histograma de les dades ha estat normalitzat al nombre de valors, de manera que el que es representa és la funció de densitat de probabilitat. La línia discontinua indica la forma que té una distribució de probabilitat gaussiana amb mitjana $\mu = 200$ i desviació típica $\sigma = 30$.

Figura 18. Densitat de probabilitat de les dades de l'exemple 3.8



La probabilitat que el senyal $x(t)$ prengui un valor en l'interval b_k estarà determinada pel quotient entre el nombre de vegades que $x(t)$ pren valors en l'interval b_k i el nombre total de valors del senyal, n . Ens referirem llavors a la *funció probabilitat* de $x(t)$ com

$$p_k = \frac{h_k}{n}. \quad (24)$$

Com caldia esperar, la suma de totes les probabilitats $\sum_{k=1}^m p_k = 1$ és un lloc que correspon a la probabilitat que el senyal adopti un valor en algun interval b_k .

La *densitat de probabilitat* $\rho_x(k)$ en un interval b_k es defineix llavors com el nombre de valors que s'han produït en aquest interval h_k normalitzat amb l'àrea total de l'histograma

$$\rho_x(k) = \frac{h_k}{n\Delta x}. \quad (25)$$

En el límit en el qual la mida de l'interval Δx es fa infinitesimalment petita (o, equivalentment, $m \rightarrow \infty$) i el nombre de valors del senyal n tendeix a infinit, $\rho_x(k)$ dóna lloc a la *funció densitat de probabilitat contínua*

$$\rho_x(x) = \lim_{\Delta x \rightarrow 0, m \rightarrow \infty} \rho_x(k). \quad (26)$$

Tornem a l'assumpte de la independència i a l'absència de correlació estadística entre dos senyals $x(t)$ i $y(t)$. De manera similar al que vam fer anteriorment, podem dividir els rangs de valors que prenen les variables x i y en m intervals de mida Δx i Δy , respectivament. Llavors podem definir la *densitat de probabilitat conjunta* $\rho_{xy}(x,y)$ de $x(t)$ i $y(t)$ a partir del nombre de vegades que els senyals x i y prenien valors en diferents intervals *de manera simultània*. Aquesta funció ens indica, per exemple, en quants instants de temps t_1, t_2, \dots, t_n el senyal x ha pres valors en el primer interval i en l'últim, la qual cosa ens permet conèixer la probabilitat que es produixin un parell de valors (x,y) alhora.

Vegeu també

L'annex d'aquest mòdul es dedica a repassar els conceptes bàsics d'estadística que són necessaris en aquest apartat.

Quan els senyals x i y són *estadísticament independents*, la densitat de probabilitat conjunta $\rho_{xy}(x,y)$ està determinada pel producte de les *densitats de probabilitat marginals* de cadascun dels senyals $\rho_x(x)$ i $\rho_y(y)$

$$\rho_{xy}(x,y) = \rho_x(x)\rho_y(y), \quad (27)$$

ja que, segons el teorema de Bayes, la probabilitat que es doni el parell de valors (x_0, y_0) és el producte de les probabilitats que x_0 es doni en x i que y_0 es doni en y de manera independent. Quan els senyals no són independents tenim, en canvi, que

$$\rho_{xy}(x,y) = \rho_x(x)\rho_y(y) + \rho_{xy}(x|y) \quad (28)$$

en què $\rho_{xy}(x|y)$ és la *probabilitat condicionada* que es doni un valor de x havent-se donat un valor y . De l'expressió anterior es deriva que els *moments centrals conjunts* de qualsevol ordre r,s compleixen la condició

$$E[(x - \bar{x})^r (y - \bar{y})^s] = E[(x - \bar{x})^r] E[(y - \bar{y})^s] \quad (29)$$

per a qualsevol valor enter de r,s . En particular, per a $r = s = 1$ tenim que la *covariància* $Cov(x,y) = E[(x - \bar{x})(y - \bar{y})]$ entre x i y és nul·la

$$Cov(x,y) = E[(x - \bar{x})(y - \bar{y})] = E[(x - \bar{x})] E[(y - \bar{y})] = 0, \quad (30)$$

ja que els primers moments centrals de cadascuna són nuls

$$E[(x - \bar{x})] = \int_{-\infty}^{\infty} (x - \bar{x})\rho_x(x)dx = \int_{-\infty}^{\infty} x\rho_x(x)dx - \bar{x} = 0 \quad (31)$$

$$E[(y - \bar{y})] = \int_{-\infty}^{\infty} (y - \bar{y})\rho_y(y)dy = \int_{-\infty}^{\infty} y\rho_y(y)dy - \bar{y} = 0. \quad (32)$$

En efecte, quan dos senyals són estadísticament independents, la seva covariància és nul·la.

A partir de la covariància es pot definir la *correlació* entre els dos senyals

$$\text{Corr}(x,y) = \frac{\text{Cov}(x,y)}{\sigma_x \sigma_y} \quad (33)$$

que dóna compte de la covariància normalitzada amb les desviacions estàndard de cadascun dels senyals σ_x i σ_y .

El codi 3.9 descriu com podem calcular les matrius de covariància i correlació entre dues variables.

Codi 3.9: Càlcul de matrius de covariància i correlació entre dos conjunts de dades

```

1 from scipy import stats
2 from numpy import *
3
4 # dades:
5 mu1, sigma1 = 200, 30
6 x = mu1 + sigma1*random.randn(3).T
7 mu2, sigma2 = 10, 5
8 y = mu2 + sigma2*random.randn(3).T
9
10 # matriu de covariància:
11 covmat = cov(x,y,bias=1)
12
13 # matriu de correlació:
14 corrmat = corrcoef(x,y)
15
16 >>> print covmat
17 [[ 902.94004704 -86.47788485]
18 [ -86.47788485  26.52406588]]
19
20 >>> print var(x), var(y)
21 902.940047043 26.5240658823
22
23 >>> print corrmat
24 [[ 1.          -0.55879891]
25 [-0.55879891  1.          ]]
26
27 # relació entre totes dues:
28 >>> cov(x/std(x),y/std(y),bias=1)
29 array([[ 1.          , -0.55879891],
30        [-0.55879891,  1.          ]])

```

Tot això ens porta a concloure que dos senyals independents tenen una correlació nul·la. El que no és necessàriament cert és que dos senyals amb correlació nul·la siguin independents. Perquè siguin independents, a més de la correlació també han de ser nuls els moments centrals conjunts de qualsevol ordre que compleixin l'equació 29.

Per exemple, per a $r = s = 2$ tenim, utilitzant l'equació 71 que

$$E[(x - \bar{x})^2(y - \bar{y})^2] = E[(x - \bar{x})^2]E[(y - \bar{y})^2] = Var[x]Var[y] \quad (34)$$

Informació mútua

La informació mútua és una mesura de la relació estadística entre dos senyals. De manera qualitativa, la informació mútua es pot entendre com una funció de correlació que té en compte relacions lineals i no lineals entre dues variables.

En concret, el que mesura la informació mútua és la probabilitat que una variable prengui un valor determinat condicionada al fet que l'altra variable n'hagi pres un altre. O el que és el mateix, la quantitat d'informació que tenim d'una en cas de conèixer l'altra.

Per a això introduïm el concepte d'*entropia de Shannon* d'un senyal x , definida com el valor esperat del logaritme de la seva densitat de probabilitat

$$I(x) = -E[\log(\rho_x(x))] = - \int \rho_x(x) \log(\rho_x(x)) dx. \quad (35)$$

Com més gran és l'entropia de Shannon d'un senyal, més petit és el seu contingut informational.

D'aquesta manera un senyal aleatori presentarà més entropia i menys contingut informational que un senyal periòdic. En l'àmbit de les comunicacions, l'equació 35 permet quantificar la informació que es transmet a través d'un determinat canal. De manera equivalent, és possible definir la *informació mú-*

tua entre dos senyals com el valor esperat del logaritme de la seva distribució conjunta

$$I(x,y) = -E[\log(\rho_{xy}(x,y))] = - \int \int \rho_{xy}(x,y) \log(\rho_{xy}(x,y)) dx dy \quad (36)$$

En el cas que x i y siguin senyals estadísticament independents, la probabilitat conjunta factoritza com el producte de les probabilitats marginals $\rho_{xy}(x,y) = \rho_x(x)\rho_y(y)$ (equació 27), i la informació mútua està determinada per la

$$I(x,y) = -E[\log(\rho_{xy}(x,y))] = -E[\log(\rho_x(x))]E[\log(\rho_y(y))] = I(x)I(y) \quad (37)$$

Quan totes dues senyals tenen alguna relació, sigui lineal o no lineal, la probabilitat condicional $\rho_{xy}(x|y)$ no és nul·la, per la qual cosa es compleix l'equació 28 i la informació mútua pren valors positius $I(x,y) = I(x)I(y) + I(x|y) > I(x)I(y)$.

ja que la definició inclou directament la densitat de probabilitat conjunta, l'entropia mútua té en compte tots els moments conjunts de les distribucions i no solament la covariància, i per tant, la informació mútua és una mesura de la independència estadística entre dos senyals.

ICA descompon una matriu de dades en components estadísticament independents.

Exemple: separació de fonts independents

Les diferents implementacions d'ICA obtenen els senyals independents d'una barreja mitjançant un procés de minimització de la informació mútua. Addicionalment, la majoria dels mètodes inclouen alguna estratègia que garanteixi que les fonts, a més d'independents, siguin el més simples possible (príncipi de parsimònia). Per a això utilitzen diferents mesures de complexitat computacional dels senyals font obtinguts.

La versió que utilitzarem està continguda en la biblioteca Python d'accés lliure MDP*, i utilitza aquest procediment que garanteix fonts independents i de baixa complexitat. La versió utilitzada implementa l'algorisme FASTICA desenvolupat per Aapo Hyvärinen.

El codi 3.10 descriu la utilització de la tècnica ICA en Python. El programa comença definint dos senyals sinusoïdals amb freqüències diferents. En l'exemple introductorí, cadascun d'aquests senyals es correspondrien a les veus de dos dels participants a la festa. Aquests senyals apareixen representats en la figura 19 i seran considerats com les fonts originals per identificar mitjançant la tècnica ICA.

*MDP (*Modular toolkit for Data Processing*)
<http://mdp-toolkit.sourceforge.net/>.

Lectura recomanada

- A. Hyvärinen.** (1999). "Fast and Robust Fixed-Point Algorithms for Independent Component Analysis". *IEEE Transactions on Neural Networks* (núm. 10, vol. 3, pàg. 626-634). **A. Hyvärinen; E. Oja** (1997). "A fast Fixed-Point Algorithms for Independent Component Analysis". *Neural Computation* (núm. 9, vol. 7, pàg. 1483-1492). <http://www.cis.hut.fi/projects/ica/fastica/>

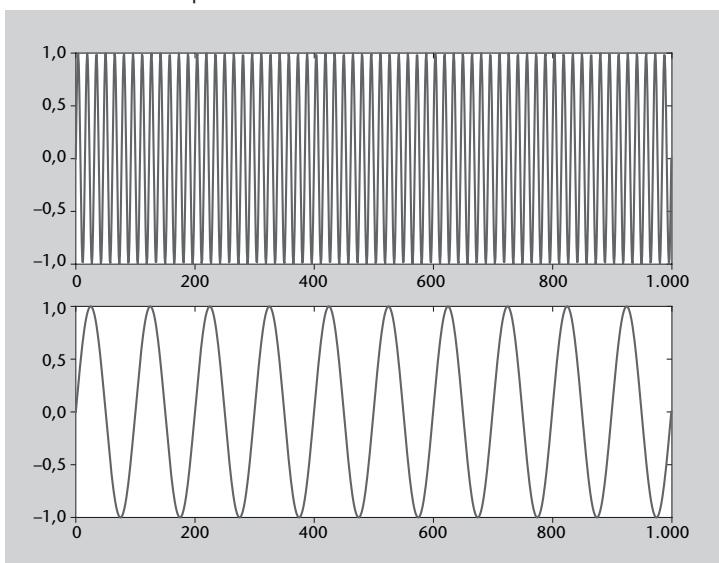
Codi 3.10: Exemple de descomposició ICA en Python

```

1 import numpy as np
2 import pylab as py
3 import scipy as sc
4 import mdp
5 from mdp import fastica
6
7 # Fonts originals
8 t = np.linspace(0,10,1000)
9 Sa = np.sin(2*sc.pi*t)
10 Sb = np.sin(13*sc.pi*t)
11
12 S1 = Sa.reshape(1,-1)
13 S0 = Sb.reshape(1,-1)
14 S = np.concatenate((S0,S1))
15
16 # Dades barrejades
17 A = [[1, 4], [-3, 2]] # Matriu de barreja
18
19 X = np.dot(A, S) # Observacions
20
21 # Descomposició ICA:
22
23 y_ICA = mdp.fastica(X.T, dtype='float32')
24
25 # representació de resultats:
26 fig1 = py.figure()
27 py.subplot(211)
28 py.plot(S.T[:,0])
29 py.subplot(212)
30 py.plot(S.T[:,1])
31 fig1.suptitle('Fonts_originals')
32
33 fig2 = py.figure()
34 py.subplot(211)
35 py.plot(X.T[:,0])
36 py.subplot(212)
37 py.plot(X.T[:,1])
38 fig2.suptitle('Fonts_barrejades')
39
40 fig3 = py.figure()
41 py.subplot(211)
42 py.plot(y_ICA[:,0])
43 py.subplot(212)
44 py.plot(y_ICA[:,1])
45 fig3.suptitle('Components_independents_(ICA)')
46
47 py.show()

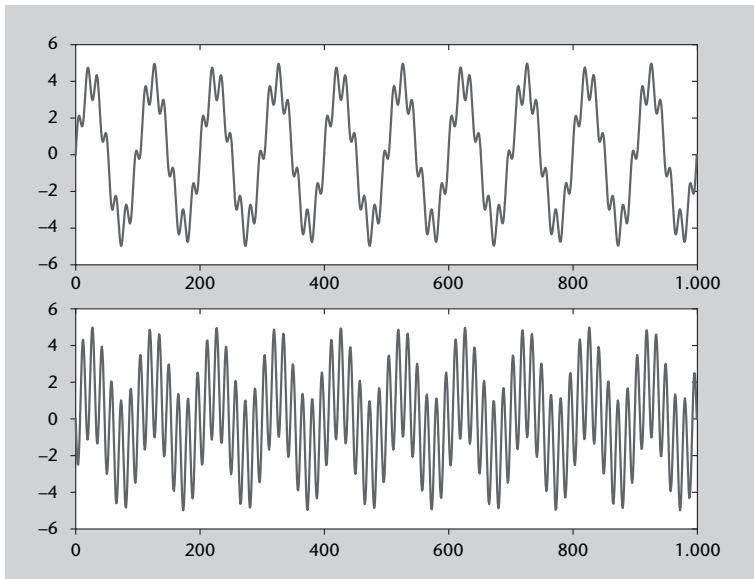
```

Figura 19. En el codi 3.10, les fonts originals són dos senyals sinusoidals amb freqüències diferents



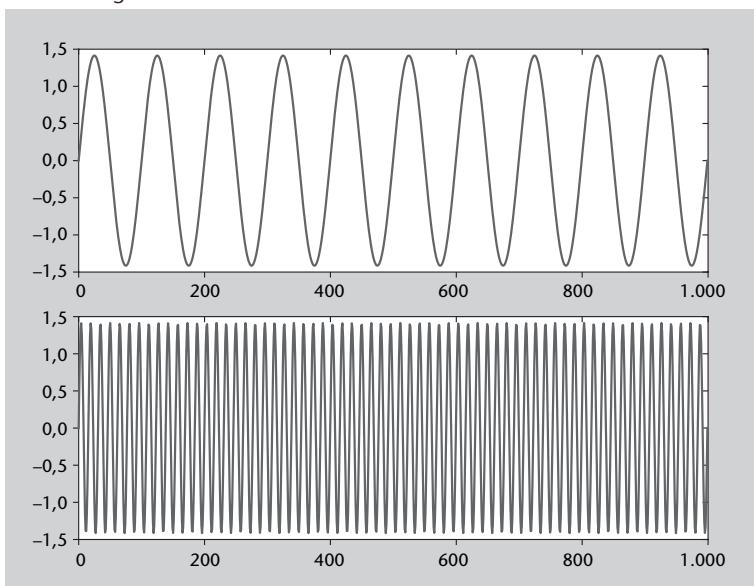
A continuació es construeixen les dades barrejades aplicant una matriu de barreges A a les fonts originals. En l'exemple de la festa, les dades barrejades corresponen al senyal total barrejat que sentiria cadascun dels assistents. Com a resultat, s'obtenen dos senyals de barreja que són combinació lineal de les fonts originals i que es representen en la figura 20.

Figura 20. Resultat de barrejar les fonts originals amb una matriu de barreja. 3.10



La descomposició ICA s'aplica a aquests dos senyals barrejats amb la intenció de poder identificar les fonts independents subjacentes. El resultat de la descomposició són els dos senyals representats en la figura 21, i com es pot veure, cadascun correspon a una de les dues fonts originals de partida. El codi es pot utilitzar per a explorar els resultats obtinguts amb la tècnica ICA en aplicar-la a un altre tipus de senyals i amb matrius de barreja diferents.

Figura 21. Els modes independents identificats per la tècnica ICA recuperen les fonts originals 3.10



3.1.4. Factorització de matrius no negatives (NMF)

Exemple d'aplicació

NMF és una eina utilitzada de manera habitual en aplicacions de *mineria de textos*, per la qual cosa l'exemple més representatiu és l'anàlisi d'aparició de paraules en textos. El mètode NMF resulta útil per a caracteritzar conjunts de dades que no prenien valors negatius, com per exemple dades de recompte (nombre de vots per candidatura, aparició de paraules en documents, freqüència d'esdeveniments en una planta industrial, etc.). Suposem que tenim un conjunt de N textos que volem caracteritzar mitjançant un subconjunt de K paraules. El sistema es pot representar mitjançant una matriu d'ocurrències A de mida $N \times K$ amb el component i,j que indica el nombre de vegades que la paraula j apareix en el text i .

$$\begin{matrix} & \text{paraula 1} & \text{paraula 2} & \dots & \text{paraula } K \\ \text{text 1} & a_{1,1} & a_{2,1} & \dots & a_{1,K} \\ \text{text 2} & a_{2,1} & a_{2,2} & \dots & a_{2,K} \\ \dots & \vdots & \vdots & \ddots & \vdots \\ \text{text } N & a_{N,1} & a_{N,2} & \dots & a_{N,K} \end{matrix}$$

Triant les K paraules de manera estratègica podrem caracteritzar els textos segons el nombre de vegades que aparegui cada paraula. Per exemple, en cas que els textos provinguin d'una web de notícies, sembla lògic que aquells textos en els quals més vegades aparegui la paraula *internacional* o *Europa* estaran referits a notícies d'àmbit internacional, mentre que aquells en què apareguin les paraules *alcalde* o *ajuntament* tindran a veure amb notícies d'àmbit local. Evidentment, també pot succeir que una notícia local sobre ajudes agràries inclogui la paraula *Europa*, o que una d'àmbit internacional es refereixi a l'alcalde d'una ciutat determinada. El mètode NMF supera aquesta ambigüïtat definint un conjunt de característiques que permeten agrupar els textos de manera eficient. En resum, NMF ens permetrà agrupar els textos seguint un criteri que els permeti identificar mitjançant l'aparició de paraules clau.

Descripció de la tècnica

NMF fa una descomposició en factors en una matriu de dades no negativa A de la manera

$$A \approx W \cdot F \quad (38)$$

en què les matrius no negatives F i W són conegudes com a *matriu de característiques* i *matriu de pesos*, respectivament. La matriu de característiques F , de

mida $M \times K$, defineix M característiques i pondera cadascuna mitjançant la importància que té cadascuna de les K paraules. Les característiques corresponen a temes genèrics que han estat definits a partir de l'agrupació de paraules en diferents textos. El component i,j de F representa la importància de la paraula j en la característica i . El seu aspecte seria el següent

$$\begin{array}{cccc} & \text{paraula 1} & \text{paraula 2} & \dots & \text{paraula } K \\ \text{característica 1} & f_{1,1} & f_{1,2} & \dots & f_{1,K} \\ \text{característica 2} & f_{2,1} & f_{2,2} & \dots & f_{2,K} \\ \dots & \vdots & \vdots & \ddots & \vdots \\ \text{característica } M & f_{M,1} & f_{M,2} & \dots & f_{M,K} \end{array}$$

D'altra banda, la matriu de pesos W atribueix un pes a cadascuna de les característiques en funció de la seva importància en cadascun dels N textos. Les M columnes corresponen a les característiques i les N files als textos analitzats, de manera que el component i,j de W indica la importància que té la característica j en el text i

$$\begin{array}{cccc} & \text{característica 1} & \text{característica 2} & \dots & \text{característica } M \\ \text{text 1} & w_{1,1} & w_{1,2} & \dots & w_{1,M} \\ \text{text 2} & w_{2,1} & w_{2,2} & \dots & w_{2,M} \\ \dots & \vdots & \vdots & \ddots & \vdots \\ \text{text } N & w_{N,1} & w_{N,2} & \dots & w_{N,M} \end{array}.$$

L'equació 38 s'ha d'interpretar com una transformació que permet definir M característiques de manera que les dades originals es puguin factoritzar de la manera

$$\text{textos} \times \text{paraules} \approx \text{textos} \times \text{característiques} \cdot \text{característiques} \times \text{paraules}. \quad (39)$$

Factorització

NMF permet factoritzar una matriu de dades definida positiva en el producte de dues matrius no negatives amb una estructura més simple.

Evidentment, un dels paràmetres que s'han de definir per a aplicar NMF és el nombre de característiques M que volem. Un valor massa alt resultaria en un etiquetatge massa específic dels textos, mentre que un valor excessivament petit agruparia els textos en unes poques característiques massa genèriques i, per tant, mancats d'informació rellevant. La matriu de característiques F és equivalent a la matriu de vectors propis que obteníem amb altres tècniques de factorització de dades com PCA o ICA. La diferència principal entre NMF i PCA és que en NMF la matriu F és no negativa i, per tant, la factorització de A es pot interpretar com una descomposició acumulativa de les dades originals.

Convé precisar que la descomposició NMF no és única, és a dir, donat un nombre de característiques M hi ha més d'una manera de descompondre les dades en la manera de l'equació 38. L'ideal és aplicar el procediment i interpretar els resultats per a obtenir informació addicional de les dades. L'algorisme que s'encarrega de fer aquesta factorització no és trivial i escapa a l'abast d'aquest llibre. En termes generals, les diferents implementacions de NMF segueixen un procés d'optimització multidimensional que minimitza el residu R de la descomposició

$$R = W \cdot F - A. \quad (40)$$

Implementació en Python

Aquí utilitzarem una implementació que es descriu en el llibre *Programming Collective Intelligence* de T. Segaran. El codi de la rutina *nmf.py* descrit és el que apareix en el codi 3.11.

Codi 3.11: Implementació d'NMF en Python descrita en el llibre *Programming Collective Intelligence*

```

1  from numpy import *
2
3  def difcost(a,b):
4      dif=0
5      # Loop over every row and column in the matrix
6      for i in range(shape(a)[0]):
7          for j in range(shape(a)[1]):
8              # Add together the differences
9              dif+=pow(a[i,j]-b[i,j],2)
10             return dif
11
12 def factorize(v,pc=10,iter=50):
13     ic=shape(v)[0]
14     fc=shape(v)[1]
15     # Initialize the weight and feature matrices with random values
16     w=matrix([[random.random() for j in range(pc)] for i in range(ic)])
17     h=matrix([[random.random() for i in range(fc)] for i in range(pc)])
18     # Perform operation a maximum of iter times
19     for i in range(iter):
20         wh=w*h
21         # Calculate the current difference
22         cost=difcost(v,wh)
23         if i%10==0: print cost
24         # Terminate if the matrix has been fully factorized
25         if cost==0: break
26         # Update feature matrix
27         hn=(transpose(w)*v)
28         hd=(transpose(w)*w*h)
29         h=matrix(array(h)*array(hn) / array(hd))
30         # Update weights matrix
31         wn=(v*transpose(h))
32         wd=(w*h*transpose(h))
33         w=matrix(array(w)*array(wn) / array(wd))
34     return w,h

```

Lectura complementària

T. Segaran (2007).
Programming Collective Intelligence. O'Reilly

El codi 3.12 presenta un exemple d'utilització de la rutina *nmf.py*. Primer es defineix una matriu de dades A , que en aquest cas conté dades aleatòries de

comptatge de 10 paraules en 100 textos. A continuació es procedeix a factoritzar la matriu A utilitzant NMF. En aquest cas hem utilitzat un total de 3 característiques. La validesa del resultat es pot analitzar comprovant el residu de l'aproximació, és a dir, les diferències entre la matriu de dades original A i l'aproximació NMF $W \cdot F$.

Codi 3.12: Exemple d'utilització del mètode NMF

```

1 from numpy import *
2 import nmf
3
4 # Matriu de dades aleatòria (100 textos x 10 paraules):
5 A = random.randint(0,100,(100,10))
6
7 #descomposició NMF utilitzant 3 característiques:
8 W,F = nmf.factorize(A,pc=3,iter=500)
9
10 print(W)
11 print(F)
12
13 # Residu de l'aproximació:
14 R = A - dot(W,F)
15
16 # màxim error relatiu de reconstrucció (%):
17 R_rel = 100*R/A
18 print(R_rel.max())

```

Altres implementacions

Hi ha altres implementacions que utilitzen algorismes d'optimització basats en el mètode de gradients projectats, com la que es pot obtenir en l'adreça web <http://www.csie.ntu.edu.tw/~cjlin/nmf/> basada en l'article següent: **C.-J. Lin.** (2007). "Projected gradient methods for non-negative matrix factorization". *Neural Computation* (núm. 19, pàg. 2756-2779).

Anàlisi de textos

Com hem comentat, una de les aplicacions més importants de la tècnica NMF és l'anàlisi de documents. Per a poder construir la matriu de recompte de paraules, primer cal fer algunes operacions de filtratge dels textos obtinguts. Suposem que disposem d'un conjunt de fitxers de text en format *.txt*. Cada text es pot llegir mitjançant la rutina Python descrita en el codi 3.13.

Codi 3.13: Lectura de fitxers de text i creació de diccionaris en Python

```

1 # -*- coding: cp1252 -*-
2 import glob
3 import numpy
4 import string
5
6 def textos(path):
7     # fer una llista de tots els fitxers amb extensió .txt del
8     # directori path:
9     filelist = glob.glob(path)
10    docs = []
11
12    # bucle sobre tots els fitxers en el directori:
13    for kfile in filelist:
14        fp = file(kfile) # obrir fitxer
15        txt = fp.read() # llegir fitxer
16        fp.close() # tancar fitxer

```

```

16
17     # filtrar caràcters:
18     txt = txt.lower() # convertir a minúscules
19     # substituir returns de carro per espais
20     txt = txt.replace('\n',' ')
21     # esborrar caràcters ASCII que no siguin l'espai
22     # (ascii 32), o les lletres a-z (ascii 97-122):
23     char_ok = [32] + range(97,122)
24     for i in range(256):
25         if i not in char_ok:
26             txt = txt.replace(chr(i),'')
27
28     # construir una llista amb les paraules del text:
29     paraules = txt.split()
30     # crear un diccionari:
31     dicc = { }
32     for ipal in range(len(paraules)):
33         pl = paraules[ipal];
34         if len(pl)>4: # excloure paraules de menys de 5 caràcters
35             if pl in dicc:
36                 dicc[pl].append(ipal) # localització de les
37                 # aparicions
38             else:
39                 dicc[pl] = [ipal]
40     # afegir nombre d'aparicions al final del diccionari
41     for ipal in range(len(paraules)):
42         pl = paraules[ipal];
43         if pl in dicc:
44             #dicc[pl].append(len(dicc[pl])) # nombre d'aparicions
45             dicc[pl].insert(0,len(dicc[pl]))
46     # agregar diccionari de cada document
47     docs.append(dicc)
48
return docs

```

El codi 3.13 utilitza un *string* d'entrada *path* en el qual s'indica el directori on estan situats els fitxers de text. En les línies 6-9 del codi genera una llista *filelist* amb els noms dels fitxers de text. El bucle de la línia 12 recorre tots els fitxers de text i executa la seqüència d'accions següents per a cadascun: obrir fitxer, llegir fitxer, tancar fitxer, eliminar tots els caràcters ASCII que siguin diferents de les lletres *a-z* o l'espai (línies 23-26). Una vegada filtrades, es construeix una llista amb les paraules que apareixen en el text mitjançant la funció Python *split*.

El pas següent és crear un diccionari Python *dicc*, que consisteix en una llista de les paraules que apareixen en el text que va acompanyada de les posicions en les quals apareix cada paraula (línies 30-38). En l'exemple només s'han incorporat al diccionari les paraules amb més de quatre caràcters. Al principi d'aquesta llista afegim un nou valor que indica el nombre de vegades que apareix cada paraula en el text, per la qual cosa cada entrada del diccionari final conté la paraula, i a continuació un vector amb el nombre de vegades que apareix en el text i les posicions del text en les quals apareix. L'última instrucció del bucle de fitxers incorpora el diccionari de cada text en una variable biblioteca *docs* que agrupa tots els diccionaris (un diccionari per a fitxer de text). La rutina retorna la variable *docs* amb tots els diccionaris de tots els textos trobats en la carpeta *path*.

El codi 3.14 indica la instrucció necessària per a generar una biblioteca de diccionaris a partir dels fitxers de text situats en la carpeta *C:/textos/*.

Codi 3.14: Crida a la rutina *textos* per a generar una biblioteca de diccionaris a partir d'un conjunt de fitxers de text

```

1 >>> docs = textos('C:\\textos\\*.txt')
2
3 >>> docs[0]
4 { 'arrhythmogenic': [3, 2, 11, 135], 'catecholaminergic': [1, 147], '
    'significant': [1, 80], 'family': [1, 52], 'providers': [1, 67], '
    'abridge': [1, 125], 'correlations': [1, 154], 'outcomes': [1, 49],
    'helped': [1, 14], 'namely': [1, 137], 'determined': [1, 100], '
    'decisions': [1, 72], 'occur': [1, 85], 'recognition': [1, 105], '
    'current': [1, 127], 'polymorphic': [1, 148], 'realm': [1, 33], '
    'knowledge': [1, 128], 'mutated': [1, 6], 'implementation': [1,
    114], 'technology': [1, 92], 'coupled': [1, 93], 'pathogenesis':
    [1, 151], 'research': [1, 1], 'better': [1, 55], 'their': [9, 8,
    7, 6, 5, 18, 48, 57, 65, 74], 'genetic': [3, 2, 35, 131], '
    'clinical': [1, 87], 'identification': [1, 4], 'treatment': [1,
    116], 'which': [1, 110], 'inherited': [3, 2, 10, 134], 'forward':
    [1, 15], 'progress': [3, 2, 26, 79], 'tachycardia': [1, 150], '
    'fatal': [1, 108], 'cause': [1, 9], 'possible': [1, 40], '
    'potentially': [1, 107], 'practice': [1, 88], 'genotypephenotype':
    [1, 153], 'diseases': [5, 4, 3, 12, 109, 136], 'advances': [1,
    90], 'genes': [1, 7], 'background': [1, 132], 'symptomatic': [1,
    44], 'understanding': [3, 2, 16, 97], 'pathophysiology': [1, 19],
    'genetically': [1, 99], 'assists': [1, 102], 'understand': [1,
    56], 'discussed': [1, 157], 'members': [1, 53], 'improving': [1,
    96], 'article': [1, 123], 'conjunction': [1, 63], 'continued': [3,
    2, 0, 78], 'earlier': [3, 2, 104, 113], 'improve': [1, 47], '
    leads': [1, 111], 'short': [1, 141], 'decades': [1, 24], 'syndrome':
    [5, 4, 3, 140, 143, 145], 'arrhythmias': [3, 2, 36, 101], '
    ventricular': [1, 149], 'patients': [1, 45], 'continue': [1, 83],
    'risks': [1, 58], 'allow': [1, 60], 'brugada': [1, 144], 'changes':
    [1, 81]}

```

En la línia 3 s'obté el diccionari corresponent al primer dels textos trobats en la carpeta. Cada entrada del diccionari consta d'un camp per a la paraula i d'un vector en el qual el primer component indica el nombre de vegades que apareix la paraula en el text, i els següents, les posicions en les quals apareix la paraula al llarg del text. Per exemple, la paraula *continued* apareix un total de tres vegades en el primer document, concretament en les paraules primera [0], tercera [2] i en la que ocupa la posició 79 [78].

La rutina 3.15 fa una llista de les 10 paraules més freqüents de cadascun dels textos analitzats. Amb aquesta informació i la biblioteca de diccionaris, podem construir la matriu d'ocurrències de paraules en textos.

Codi 3.15:

```

1 from textos import *
2 import numpy
3 import operator
4
5
6 # obtenir un conjunt de diccionaris:
7 docs = textos('C:\\textos\\*.txt')
8
9 # calcular la matriu de recompte de paraules
10 # mida N textos x K paraules
11
12 dicc_sort = []
13 # loop over dictionaries
14 for k in range(len(docs)):
15     # ordenar diccionari per nombre d'aparicions
16     ds = sorted(docs[k].iteritems(), key=operator.itemgetter(1), reverse=
        True)

```

```

17     dicc_sort.append(ds)
18 # Seleccionar un grup de K paraules que
19 # apareguin en els diferents diccionaris:
20     print k
21 # fer una llista de les 10 paraules més utilitzades en cada text:
22     for kw in range(1,10):
23         print ds[kw][0]

```

Considerem una matriu d'ocurrències de $K = 5$ paraules en $N = 4$ textos com la següent:

$$A = \begin{matrix} & \text{paraula 1} & \text{paraula 2} & \text{paraula 3} & \text{paraula 4} & \text{paraula 5} \\ \text{text 1} & 14 & 1 & 3 & 2 & 1 \\ \text{text 2} & 1 & 2 & 0 & 14 & 1 \\ \text{text 3} & 15 & 1 & 3 & 2 & 4 \\ \text{text 4} & 0 & 1 & 3 & 21 & 0 \end{matrix}$$

En aquest cas la primera paraula apareix molt en els textos 1 i 3 i poc en els altres, mentre que els textos 2 i 4 comparteixen una alta aparició de la paraula 4. Les matrius F, W resultants d'aplicar el codi 3.11 a la matriu anterior s'indiquen en el codi 3.16.

Codi 3.16:

```

1 >>> F
2 matrix ([[ 1.56842028e+01,   1.10030496e+00,   3.18461101e+00,
3           2.07222144e+00,   2.79548846e+00],
4           [ 8.29519007e-14,   6.26129490e-01,   7.91626659e-01,
5           8.31254256e+00,   1.45667562e-01]])
6 >>> W
7 matrix ([[ 8.78583342e-01,   2.08132147e-02,
8           [ 5.58125546e-02,   1.66257600e+00],
9           [ 9.69522495e-01,   1.99310161e-05],
10          [ 5.33686016e-03,   2.53004685e+00]])

```

La interpretació dels resultats és la següent: la matriu F té dues files que corresponen a cadascuna de les dues característiques que s'han extret per la tècnica NMF. En aquesta matriu F , les columnes corresponen a cadascuna de les paraules. Analitzant els valors de F que apareixen en el codi 3.16, la primera conclusió a la qual arribem és que la primera característica té a veure principalment amb la primera paraula ja que el component $F(1,1)$ de la matriu pren un valor elevat. El mateix succeeix amb la segona característica, que apareix relacionada principalment a la quarta paraula (component $F(2,4)$ de la matriu F). Per tant, la matriu F ens descriu dues característiques que permeten analitzar els textos, una relacionada amb la paraula 1 i una altra amb la paraula 4. Aquesta informació, de fet, és rellevant, ja que posa de manifest algunes de les característiques de la matriu d'ocurrències inicial.

La matriu W , al seu torn, pondera la importància de cadascuna de les dues característiques en els quatre textos analitzats. La característica 1 (primera columna de W), per exemple, adopta valors més alts en els textos 1 i 3 (com-

ponents $W(1,1)$ i $W(3,1)$). De manera similar, la segona característica (relacionada amb la paraula 4), permet caracteritzar els textos 2 i 4 (components $W(2,2)$ i $W(4,2)$).

En resum, la descomposició NMF ens ha permès trobar dues característiques que permeten agrupar els textos segons els patrons d'aparició de paraules en cadascun. En aquest exemple les característiques estan relacionades principalment amb una de les paraules, però en general seran una combinació lineal d'algunes de les paraules analitzades. Convé recordar que la inicialització aleatòria de les matrius W, F que apareix en el codi 3.11 ocasiona que els resultats obtinguts siguin lleugerament diferents cada vegada que s'executa el codi. També cal notar que els resultats poden ser millorats mitjançant l'ajust de paràmetres com el nombre màxim d'iteracions.

3.2. Discriminació de dades en classes

En els subapartats anteriors hem vist que la tècnica PCA serveix fonamentalment per a *caracteritzar* un conjunt de dades a partir de les variables que presenten més variabilitat. D'igual manera, ICA s'utilitza per a *identificar* els diferents mecanismes subjacents que donen lloc a l'estructura de les dades. L'anàlisi de discriminantes serveix per a determinar quines variables d'un conjunt de dades són les que discriminen entre dues o més classes predefinides.

En termes generals, es pot dir que PCA és útil per a caracteritzar, ICA per a identificar i LDA per a *discriminar*.

Aquí ens limitarem al cas en el qual es pretenguin distingir dues classes diferents, per la qual cosa ho tractarem com a mètode de *dicotomia*. Ens limitarem al cas de dues classes perquè les expressions matemàtiques són més simples i permeten una comprensió més directa de la tècnica. L'estensió de la tècnica a casos en els quals calgui distingir més de dos grups no representa una dificultat addicional excessiva.

Dicotomia segons la RAE

Segons el diccionari de la Real Acadèmia Espanyola *dicotomia* és el mètode de classificació en què les divisions i subdivisions solament tenen dues parts.

3.2.1. Anàlisi de discriminants lineals (LDA)

Exemple d'aplicació

És freqüent que un conjunt de dades estigui compost per observacions que puguin ser classificades en dos grups. Per exemple, imaginem una base de dades amb mesures biomecàniques d'atletes fetes durant l'execució d'un exercici de salt d'alçada. Resultaria interessant saber quines variables biomecàniques dels

saltadors permeten discriminar entre els atletes que assoleixen el podi de la resta de participants. La base de dades inclourà mesures de diverses magnituds rellevants com la velocitat de sortida, la longitud de la primera gambada o la inclinació del cos en fer l'últim pas. Aquestes n variables biomecàniques es mesuraran per a un conjunt de m saltadors durant campionats internacionals. Els mesuraments donaran lloc a un conjunt de n variables i m observacions, i per tant, a una matriu de dades de mida $m \times n$. A les observacions haurem d'afegir un vector d'etiquetes que identifiqui cada observació com a grup 1 si el saltador va ocupar el podi o com a grup 2 si va finalitzar el campionat a partir de la quarta posició.

El nostre interès és saber quines variables o combinació d'aquestes ens permeten identificar als saltadors que ocupen el podi en competicions internacionals. Una anàlisi PCA d'aquestes dades ens permetria identificar les variables que expressin millor la variabilitat entre saltadors, però aquestes no són necessàriament les que *discriminaran* millor entre els saltadors que presenten un millor rendiment. Pot succeir, per exemple, que la màxima variabilitat es presenta en la manera com es fa el primer pas sense que aquesta variable tingui un impacte decisiu en l'alçada final superada pel saltador, i per tant, en el fet d'assolir el podi.

Descripció de la tècnica

El mètode LDA parteix d'un conjunt de dades, que anomenarem *conjunt d'entrenament*, del qual tenim un coneixement previ de l'assignació de cadascuna de les observacions a un grup. En l'exemple dels saltadors d'alçada, el conjunt d'entrenament consistiria en un conjunt de m observacions de n variables biomecàniques durant una execució del salt. Això ens permet construir un conjunt de dimensió n de dades (n variables, m observacions) als quals s'ha assignat una etiqueta de pertinença a un dels dos grups (p. ex., etiquetes que identifiquen els m saltadors com del grup 1 (podi) o del grup 2 (fora del podi)).

Com en nombroses tècniques d'intel·ligència artificial, l'LDA es compon de dues fases: una primera d'*entrenament* i una segona de *predicció*. En la fase d'entrenament es parteix de les m dades del conjunt d'entrenament i es genera un model lineal de les n variables que permeti discriminar entre els dos grups. En la fase de predicció, el model lineal s'utilitza per a predir a quin grup pertany una nova observació que no hagi estat utilitzada durant la fase d'entrenament.

En l'exemple anterior, la predicció ens permetria estimar si un nou saltador serà candidat al podi (pertany al grup 1) o no (grup 2) a partir d'una mesura de les seves n variables biomecàniques.

La formulació matemàtica de la tècnica LDA implica a modelitzar probabilísticament la distribució de dades en cadascuna de les classes. Nosaltres ens limitem al cas de dues classes, a les quals ens referirem com α_1, α_2 . Durant la fase d'entrenament, les dades d'entrenament de cada classe s'utilitzen per a fer una estimació estadística dels paràmetres del model (mitjana, variància). El procés d'entrenament consisteix, per tant, en un procés d'*estimació estadística*, en el qual es determinen paràmetres d'una certa densitat de probabilitat.

La versió més simple de l'anàlisi de discriminants assumeix que les dades de les classes α_1, α_2 estan repartides seguint una distribució de probabilitat normal. Això s'expressa mitjançant la densitat de probabilitat condicionada que una dada \mathbf{x} pertanyi a la classe α_i

$$p(\mathbf{x}|\alpha_i) = \frac{1}{(2\pi)^{N/2}\sigma_i} \exp\left\{-\frac{1}{2\sigma_i^2}(\mathbf{x} - \mathbf{m}_i)^T \cdot (\mathbf{x} - \mathbf{m}_i)\right\}, \quad (41)$$

en què N és la dimensionalitat de les dades (nombre de variables), i μ_i i σ_i són la mitjana i desviació típica de les dades pertanyents a la classe α_i .

Una vegada modelitzades les dades, s'introduceix una *funció discriminant* $g(\mathbf{x})$ que permet assignar la dada \mathbf{x} a una de les dues classes. L'assignació d'una nova dada \mathbf{x}_0 es fa de la manera següent: si $g(\mathbf{x}_0) > 0$, la dada \mathbf{x}_0 s'assigna a la classe α_1 , mentre que si $g(\mathbf{x}_0) < 0$ serà assignat al grup α_1 . En cas que la funció discriminant sigui zero $g(\mathbf{x}_0) = 0$, el valor \mathbf{x}_0 es trobaria en la frontera de decisió i serà igualment assignable a qualsevol de les classes.

La funció discriminant $g(\mathbf{x})$ permet definir la frontera de decisió de les dades per a assignar-les a una classe. En el cas en què tinguem dues classes per discriminar, la funció discriminant estaria determinada per

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}), \quad (42)$$

en què $g_i(\mathbf{x})$ és la funció d'assignació a la classe i , que està determinada per l'expressió

$$g_i(\mathbf{x}) = \ln p(\mathbf{x} | \alpha_i) + \ln P(\alpha_i), \quad i = 1, 2 \quad (43)$$

En l'equació anterior, $p(\mathbf{x} | \alpha_i)$ és la densitat de probabilitat condicionada que una dada \mathbf{x} pertanyi a la classe α_i descrita en l'equació 41, i $P(\alpha_i)$ és la *densitat de probabilitat a priori* a la qual fèiem referència anteriorment. $P(\alpha_i)$ és, per tant, la probabilitat amb la qual assignaríem *a priori* una dada a la classe α_i . Substituint l'expressió 41 en l'equació 43 i introduint aquesta al seu torn en 42 és fàcil veure que la funció discriminant de l'equació està determinada per

$$g(\mathbf{x}) = -\frac{1}{\sigma_1^2} (\mathbf{x} - \mathbf{m}_1)^T \cdot (\mathbf{x} - \mathbf{m}_1) + \frac{1}{\sigma_2^2} (\mathbf{x} - \mathbf{m}_2)^T \cdot (\mathbf{x} - \mathbf{m}_2) - N \ln \frac{\sigma_1}{\sigma_2} + \ln P(\alpha_1) - \ln P(\alpha_2). \quad (44)$$

En el cas en el qual no hi hagi informació *a priori* sobre la pertinença a una de les dues classes, tindrem $P(\alpha_1) = P(\alpha_2) = \frac{1}{2}$ i la funció discriminant se simplifica i adopta la forma

$$g(\mathbf{x}) = -\frac{1}{\sigma_1^2} (\mathbf{x} - \mathbf{m}_1)^T \cdot (\mathbf{x} - \mathbf{m}_1) + \frac{1}{\sigma_2^2} (\mathbf{x} - \mathbf{m}_2)^T \cdot (\mathbf{x} - \mathbf{m}_2) - N \ln \frac{\sigma_1}{\sigma_2}. \quad (45)$$

L'equació 45 es pot interpretar de manera geomètrica: l'assignació de la dada \mathbf{x} a una o una altra classe implica calcular a quina distància es troba la dada \mathbf{x} del centroide de cada classe i , que està determinada per

$$d_i(\mathbf{x}) = \frac{1}{\sigma_i^2} (\mathbf{x} - \mathbf{m}_i)^T \cdot (\mathbf{x} - \mathbf{m}_i). \quad (46)$$

La distància $d_i(\mathbf{x})$ expressada en l'equació anterior és coneguda com a *distància de Mahalanobis*, i constitueix una mètrica que pondera la distància de la dada al centroide de la classe i \mathbf{m}_i utilitzant la variància de les dades de la classe i , σ_i^2 . La funció discriminant $g(\mathbf{x})$ està composta per les distàncies de Mahalanobis de la dada a cadascuna de les classes, i inclou una correcció deguda a la possible disparitat entre les variàncies de les dades de cada classe $-N \ln \frac{\sigma_1}{\sigma_2}$. Així, per a assignar una dada \mathbf{x}_0 a la classe α_1 , s'haurà de complir $g(\mathbf{x}_0) > 0$, o el que és el mateix, que $g_1(\mathbf{x}_0) > g_2(\mathbf{x}_0)$, i per tant

$$\frac{1}{\sigma_1^2} (\mathbf{x}_0 - \mathbf{m}_1)^T \cdot (\mathbf{x}_0 - \mathbf{m}_1) + N \ln \sigma_1 < \frac{1}{\sigma_2^2} (\mathbf{x}_0 - \mathbf{m}_2)^T \cdot (\mathbf{x}_0 - \mathbf{m}_2) + N \ln \sigma_2. \quad (47)$$

Segons l'equació 47, quan les variàncies de tots dos grups siguin iguals $\sigma_1 = \sigma_2$, el discriminador lineal LDA assignarà la dada \mathbf{x}_0 a la classe el centroide de la qual estigui més a prop de la dada en distància Mahalanobis. En aquest cas, la frontera de decisió serà equidistant en distància Mahalanobis als centroides dels dos grups. En cas que hi hagi una discrepància entre les dues desviacions típiques (p. ex., $\sigma_1 \neq \sigma_2$), el classificador penalitzarà l'assignació al grup que presenti més dispersió estadística mitjançant el terme $N \ln \sigma_i, i = 1, 2$, la qual cosa ocasionarà un desplaçament efectiu de la frontera de decisió del problema. A més de la interpretació geomètrica de l'assignació de pertinença a classes, el mètode es pot entendre des d'un punt de vista estadístic, ja que la nova dada és assignada al clúster amb pertinença *estadísticament més versemblant*.

Funció de versemblança

Moltes altres tècniques estadístiques de reconeixement de patrons utilitzen la funció de versemblança per a definir els criteris d'agrupament.

Exemple: separació de dades bidimensionals

Les dues fases de la tècnica LDA es descriuen en l'exemple descrit en el codi 3.17. Es parteix d'un conjunt d'entrenament amb dades gaussianes ($n = 2$

variables, $m = 100$ observacions) que han estat etiquetades com a pertanyents al grup 1 o al grup 2. Les dades etiquetades com a grup 1 tenen la mateixa desviació típica que les del grup 2 ($\sigma_1 = \sigma_2 = 5,5$), però presenten una mitjana diferent ($\mu_1 = 10$, $\mu_2 = -10$). El conjunt complet de dades, al costat de les etiquetes de pertinença a cada grup, es passa a la rutina LDA perquè construeixi un model lineal que permeti discriminar entre totes dues classes. En aquest exemple utilitzarem el codi LDA del paquet Python d'intel·ligència artificial *Scikit-Learn*, que podem obtenir a <http://scikit-learn.sourceforge.net>. Aquest paquet utilitza al seu torn els paquets estàndard de computació i representació gràfica de Python *numpy*, *scipy* i *matplotlib*. La fase d'entrenament correspon a les línies 26-28, en les quals es genera un model lineal (*fit*) a partir de les dades XT i les seves etiquetes de pertinença respectives als dos grups, emmagatzemades en la variable *labelT*. L'opció *priors = None* de la rutina LDA especifica que no es considera més probabilitat d'assignació *a priori* de la nova dada a una de les dues classes. De vegades, un coneixement previ de les dades ens permet afavorir l'assignació d'una nova dada a un dels grups. Aquest biaix anticipat s'expressa mitjançant la *densitat de probabilitat a priori*, que determina una certa probabilitat d'assignació als grups abans de conèixer la nova dada per classificar.

Codi 3.17: Exemple d'aplicació de l'anàlisi de discriminants lineals (LDA)

```

1 import numpy as np
2 import pylab
3 from scikits.learn.lda import LDA
4
5 # Dades gaussianes:
6 mu1, sigma1 = 10, 5.5
7 X1 = mu1 + sigma1*np.random.randn(100,2)
8 mu2, sigma2 = -10, 5.5
9 X2 = mu2 + sigma2*np.random.randn(100,2)
10
11 # Representar gràficament les dades d'entrenament:
12 fig1 = pylab.figure()
13 pylab.scatter(X1[:,0],X1[:,1],marker ='^',c='r')
14 pylab.scatter(X2[:,0],X2[:,1],marker ='o',c='b',hold='on')
15 pylab.legend(('Grupo_1', 'Grupo_2'))
16
17 # Concatenar els dos conjunts de punts:
18 XT = np.concatenate((X1,X2))
19
20 # Etiquetar les dades com a tipus 1 o tipus 2:
21 label1 = np.ones(X1.shape[0])
22 label2 = 2*np.ones(X2.shape[0])
23 labelT = np.concatenate((label1,label2))
24
25 # Fase d'entrenament:
26 clf = LDA()
27 clf.fit(XT, labelT)
28 LDA(priors=None)
29
30 #Fase de prediccio:
31 print clf.predict([[20, 0]]) #predicció per a la dada [20,0]
32 print clf.predict([[5, -20]])#predicció per a la dada [5,-20]
33
34 # Representació de la predicció de les dades [20,0] i [5,-20]:
35 pylab.scatter(20,0,s=100,marker ='x',c='k')
36 pylab.annotate('LDA_grup_1',xy=(20,-2),xycoords='data',
37 xytext=(-50,-50),
38 textcoords='offset_points',
39 arrowprops=dict(arrowstyle="->"))

```

```

40 pylab.scatter(-15,-5,s=100,marker = 'x',c='k')
41 pylab.annotate('LDA_grupo_2',xy=(-15,-5),xycoords='data',
42                 xytext=(-50,50),
43                 textcoords='offset_points',
44                 arrowprops=dict(arrowstyle="->"))
45
46 # Predicció de dades en una retícula:
47 fig2 = pylab.figure()
48 for i in range(-20,21,1):
49     for k in range(-20,21,1):
50         p = clf.predict([[i, k]])
51         print i,k,p
52         if p == 1:
53             pylab.scatter(i,k,s=20,marker='o',c='r',hold='on')
54         else:
55             pylab.scatter(i,k,s=20,marker = 'o',c='b',hold='on')
56
57 pylab.axis([-20,20,-20,20])
58 pylab.text(5,5,'GRUP_1',fontsize=25,fontweight='bold',color='k')
59 pylab.text(-10,-10,'GRUP_2',fontsize=25,fontweight='bold',color='k')
60
61 pylab.show()

```

La fase de predicción permet utilitzar el model lineal construït després de l'entrenament per a assignar una nova dada a un dels dos grups. En el codi d'exemple, les noves dades són (20,0) i (5, -20), i són assignades pel classificador lineal LDA als grups 1 i 2, respectivament (línies 31 i 32). El resultat d'aquesta assignació es representa en la figura 22, en la qual les dades d'entrenament són els cercles blaus (etiquetats com a grup 1) i els triangles vermells (grup 2). Els dues noves dades (20,0) i (5, -20) per a les quals es fa la predicción s'indiquen en la figura 22 amb una creu, i el mètode LDA les ha assignat correctament als grups 1 i 2, respectivament. La frontera de decisió que determina l'assignació de les dades a cada grup es pot obtenir calculant la predicción LDA dels punts d'un reticle de mida (-20,20) en totes dues variables.

Figura 22. Representació gràfica de la fase d'entrenament de l'exemple LDA 3.17

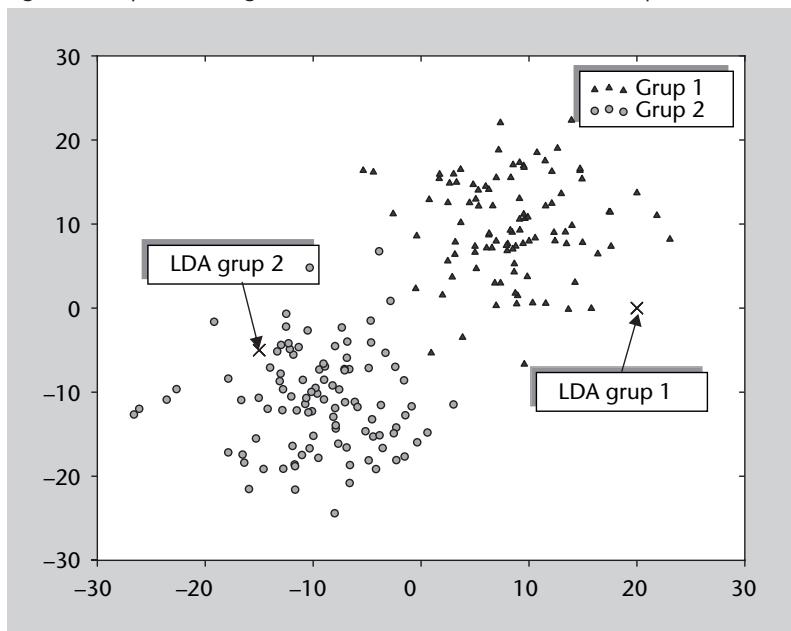


Figura 22

Els cercles blaus han estat etiquetats com a grup 1 i els triangles vermells com a grup 2 (dades d'entrenament). Després de l'entrenament, les noves dades [20,0] i [5, -20] són correctament assignades pel classificador lineal LDA als grups 1 i 2, respectivament.

En la figura 23 els punts del reticle que han estat assignats al grup 1 es representen en blau, mentre que els que han estat assignats al grup 2 es representen en vermell. Com cal esperar, la frontera és una funció lineal que separa de manera equidistant els centroides de tots dos grups. Com veurem més endavant, la frontera és equidistant a causa que la desviació típica de les dades de cada-
cun dels grups és la mateixa (és a dir, $\sigma_1 = \sigma_2$).

Figura 23

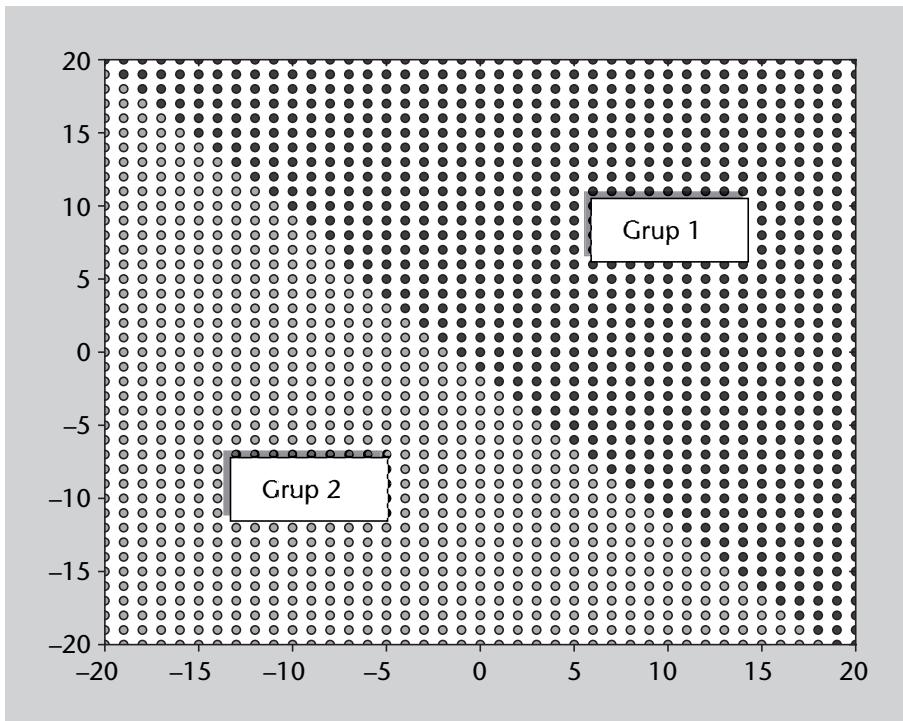
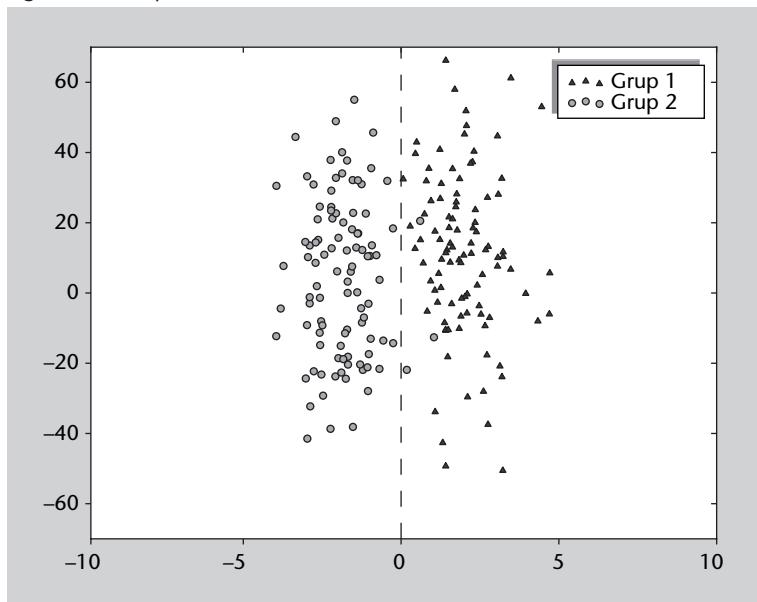


Figura 23

Predictió de l'assignació a les dues classes mitjançant el model lineal generat per la tècnica LDA a partir de les dades d'entrenament en l'exemple 3.17.

Convé aclarir que, en molts casos, tècniques com la PCA no són útils per a fer discriminació de dades en classes. Com a exemple, considerem les dades gaussianes multivariades dels dos grups de la figura 24. Una anàlisi PCA de les dades de totes dues classes proporcionaria un component principal PCA al llarg de l'eix d'ordenades (y), ja que aquesta és la direcció, i les dades presenten més variabilitat. Si observem la distribució de les dades de totes dues classes, resulta evident que una variable d'aquest tipus no resulta útil per a poder diferenciar les dades de tots dos grups, ja que les diferències entre el grup 1 i el 2 són la seva ubicació en diferents posicions de l'eix d'abscisses (x). En canvi, la línia vertical discontinua correspon a la funció discriminant obtinguda en aplicar LDA a les dades, que sí que resulta eficaç per a separar de manera automàtica les dades en dos grups, que en gran mesura coincideixen amb els grups definits inicialment. En aquest cas, totes les dades del grup 1 serien classificades correctament per la funció discriminant, mentre que només tres punts del grup 2 serien assignats erròniament al grup 1.

Figura 24. Comparació entre LDA i PCA



3.3. Visualització de dades multidimensionals

3.3.1. Escalament multidimensional (MDS)

Exemple d'aplicació

De vegades volem visualitzar dades multidimensionals en un pla. Imaginem, per exemple, un conjunt de dades que contingui múltiples variables sobre les preferències musicals d'un conjunt de persones. Les preferències individuals de cada persona es representen mitjançant un conjunt de dades que defineix nombrosos aspectes mitjançant n variables de caràcter musical (grups favorits, l'estil de música, compositors, solistes, orquestres, enregistraments, etc.) i sociològic (edat, gènere, formació, salari, etc.). A les discogràfiques els resultarà útil saber si hi ha relacions entre els consumidors habituals dels seus productes i altres col·lectius desconeguts amb els quals es pugui establir connexions.

Resulta evident que per a establir aquestes relacions de manera manual no és possible treballar en l'espai de dimensió n . Una de les possibilitats és projectar les dades de l'espai original a un espai de dimensió 2 (pla), però de manera que es mantinguin les distàncies relatives entre cadascun dels individus. En altres paraules, persones que estaven lluny en l'espai original de dimensió n també estaran lluny en la projecció bidimensional de les dades, mentre que persones properes en l'espai original ho continuaran essent en l'espai projectat. Si les distàncies entre individus es conserven en el pla, serà molt fàcil veure quines persones estan properes en l'espai original i, per tant, qui és susceptible d'adquirir un producte determinat, al qual podrien ser afins. Podria succeir, per

exemple, que els amants incondicionals del rock simfònic i els Rolling Stones, més grans de 40 anys i amb un alt nivell d'estudis, siguin un bon objectiu de mercat per a un enregistrament inèdit d'una sonata de F. Schubert del pianista rus S. Richter. Aquest tipus de relacions permet establir nous nínxols de mercat o explorar noves possibilitats en un estudi de màrqueting.

Descripció de la tècnica

Com hem indicat, l'escalament multidimensional* s'utilitza principalment com a eina de visualització de dades. En general, una anàlisi exploratòria prèvia és enormement útil per a establir relacions inicials entre les variables, i per tant, per a obtenir informació qualitativa sobre les característiques estructurals de les dades. Quan el conjunt de dades multidimensionals té un elevat nombre de variables, l'anàlisi exploratòria es complica a causa del nombre de variables per considerar. Per descomptat, sempre és possible visualitzar les dades mitjançant projeccions bidimensionals o tridimensionals en subespais de variables representatives, però el procediment no deixa de ser laboriós i dificulta una visió general de les dades. L'MDS és una eina que permet projectar les dades en un espai de dimensionalitat reduïda, de manera que la distribució de les dades en l'espai projectat mantingui una similitud amb la distribució de les dades en l'espai original.

*En anglès *multidimensional scaling (MDS)*.

En resum, l'MDS busca un espai de dimensionalitat reduïda en el qual es mantinguin les distàncies relatives entre les dades originals. Punts propers en l'espai original són escalats de manera que també siguin propers en l'espai MDS, cosa que ens permetrà una visualització millor de les dades sense desvirtuar-ne l'estructura original.

L'estructura general de l'algorisme MDS és la següent (codi 3.18): en primer lloc es prenen les dades en l'espai N -dimensional original (N variables, M observacions) i es calcula la *matriu de distàncies* entre cadascuna de les M observacions. Això dóna lloc a una matriu simètrica $M \times M$, el component de la qual i,j correspon a la distància euclidiana entre els punts \mathbf{x}_i i \mathbf{x}_j . A continuació, les M observacions són distribuïdes aleatoriament en un espai bidimensional. Per a cada parell de punts, es calcula una funció error com la diferència entre la distància entre els punts en l'espai original i la distància en l'espai projectat. Llavors s'aplica un procés iteratiu, en el qual la posició de cada punt en l'espai 2D és corregida per a minimitzar l'error entre la distància en l'espai original. El procés continua fins que l'error global no es veu afectat de manera substancial per nous desplaçaments dels nodes.

L'escalament multidimensional (MDS) permet visualitzar dades multidimensionals en un espai bidimensional.

Codi 3.18: Codi MDS en el llibre *Programming Collective Intelligence*

```

1  from numpy import *
2  from pearson import *
3  from euclidean import *
4
5  def scaledown(data, distance=euclidean, rate=0.01):
6      n=len(data)
7      # The real distances between every pair of items
8      realdist=[[distance(data[i],data[j]) for j in range(n)]
9                 for i in range(0,n)]
10    outersum=0.0
11     # Randomly initialize the starting points of the locations in 2D
12     loc=[[random.random(),random.random()] for i in range(n)]
13     fakedist=[[0.0 for j in range(n)] for i in range(n)]
14     lasterror=None
15     for m in range(0,1000):
16         # Find projected distances
17         for i in range(n):
18             for j in range(n):
19                 fakedist[i][j]=sqrt(sum([pow(loc[i][x]-loc[j][x],2)
20                                         for x in range(len(loc[i]))]))
21         # Move points
22         grad=[[0.0,0.0] for i in range(n)]
23         totalerror=0
24         for k in range(n):
25             for j in range(n):
26                 if j==k: continue
27                 # The error is percent difference between the distances
28                 errorterm=(fakedist[j][k]-realdist[j][k])/realdist[j][k]
29                 # Each point needs to be moved away from or towards the other
30                 # point in proportion to how much error it has
31                 grad[k][0]+=((loc[k][0]-loc[j][0])/fakedist[j][k])*errorterm
32                 grad[k][1]+=((loc[k][1]-loc[j][1])/fakedist[j][k])*errorterm
33                 # Keep track of the total error
34                 totalerror+=abs(errorterm)
35         print totalerror
36         # If the answer got worse by moving the points, we are done
37         if lasterror and lasterror<totalerror: break
38         lasterror=totalerror
39         # Move each of the points by the learning rate times the gradient
40         for k in range(n):
41             loc[k][0]-=rate*grad[k][0]
42             loc[k][1]-=rate*grad[k][1]
43     return loc

```

La rutina descrita en el codi 3.18 correspon a la implementació d'MDS descrita en el llibre *Programming Collective Intelligence*, T. Segaran (O'Reilly, 2007).

Exemple: projecció 2D de dades multidimensionals. Utilització de diferents mètriques

Un exemple d'utilització de la rutina es descriu en el codi 3.19. En aquest exemple es generen 30 dades, 4 dimensionals, aleatoriament distribuïdes en tres grups diferents de mida igual (línies 1-15). A continuació es crida la rutina MDS (línia 23), i prèviament es defineix la mètrica amb la qual volem treballar per a determinar la distància entre punts. En aquest exemple s'ha escollit una distància euclidiana, ja que en l'espai original les dades no presenten cap correlació. L'opció de distància de Pearson resulta interessant en casos en els quals les dades no es distribueixen per grups separats en l'espai sinó per la presència de correlacions internes entre grups de dades.

Codi 3.19: MDS amb dades descorrelacionades distribuïdes en un espai

```

1 from numpy import *
2 from scaledown import *
3 import pylab as py
4
5 # Matriu de dades (N variables x M observacions):
6 N = 4
7 M = 30
8
9 # Generar dades a partir de 3 clústers diferents:
10
11 X1 = 10 + 2*random.randn(M/3,N) # clúster 1 (dispersió mitjana)
12 X2 = -10 + 5*random.randn(M/3,N) # clúster 2 (dispersió alta)
13 X3 = 1*random.randn(M/3,N) # clúster 3 (dispersió baixa)
14
15 A = concatenate((X1,X2,X3))
16
17 # Escollir mètrica d'espai original (Euclides o Pearson):
18 metrica = pearson
19 # metrica = euclidean
20
21 # Tècnica MDS: retorna la posició de les dades
22 # originals en un espai 2D:
23 mds = scaledown(A,metrica)
24
25 # Representació de les dades en l'espai 2D MDS:
26 fig1 = py.figure()
27 for i in range(0,M/3,1):
28     py.scatter(mds[i][0],mds[i][1],marker='o',c='r')
29 for i in range(M/3,2*M/3,1):
30     py.scatter(mds[i][0],mds[i][1],marker='v',c='g')
31 for i in range(2*M/3,M,1):
32     py.scatter(mds[i][0],mds[i][1],marker='x',c='b')
33 fig1.subtitle('Metrica_Pearson')
34 py.show()

```

L'algorisme per a calcular la distància de correlació de Pearson es descriu en el codi 3.20. Aquesta distància pren un valor 1 quan les dades estan completament correlacionades i un valor 0 quan no hi ha cap correlació entre les observacions.

Codi 3.20: Distància de correlació de Pearson

```

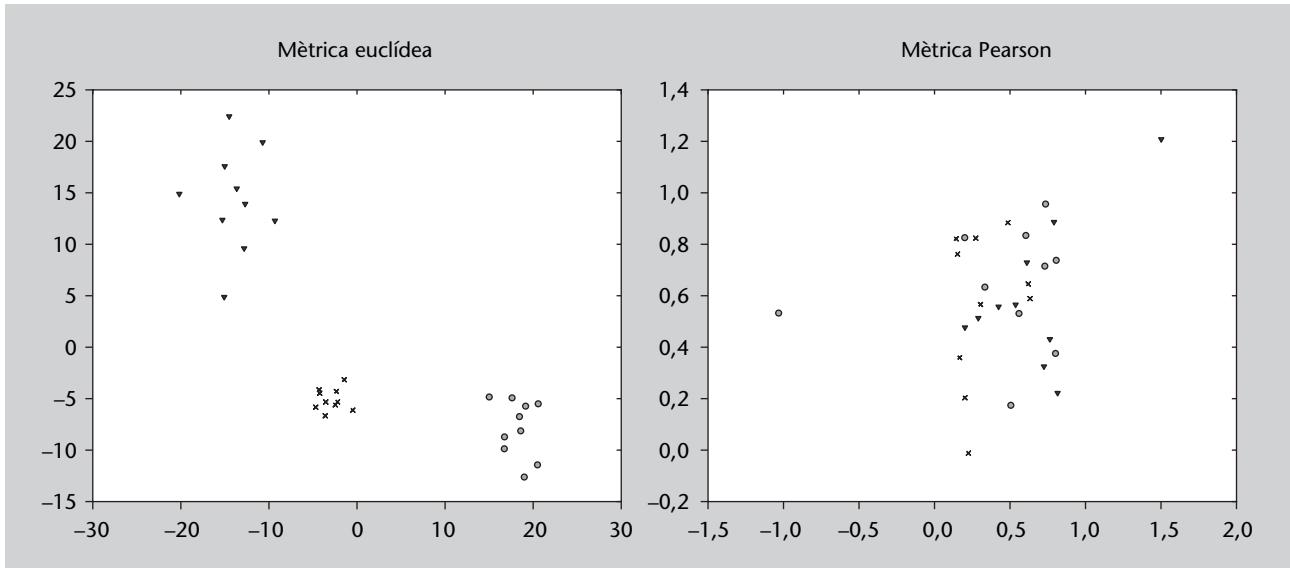
1 from math import sqrt
2
3 def pearson(v1,v2):
4     # Simple sums
5     sum1=sum(v1)
6     sum2=sum(v2)
7     # Sums of the squares
8     sum1Sq=sum([pow(v,2) for v in v1])
9     sum2Sq=sum([pow(v,2) for v in v2])
10    # Sum of the products
11    pSum=sum([v1[i]*v2[i] for i in range(len(v1))])
12    # Calculate r (Pearson score)
13    num=pSum-(sum1*sum2/len(v1))
14    den=sqrt((sum1Sq-pow(sum1,2)/len(v1))*(sum2Sq-pow(sum2,2)/len(v1)))
15    if den==0: return 0
16    return 1.0-num/den

```

La projecció bidimensional MDS de les dades descrites en l'exemple 3.19 utilitzant una mètrica euclidiana es representa en el panell esquerre de la figura 25.

Com es pot observar, els tres grups definits en l'espai original de quatre dimensions es mantenen agrupats en la projecció MDS 2D. Si utilitzem la distància de correlació de Pearson, els resultats no són satisfactoris, ja que la distribució de les dades en l'espai projectat no reflecteix l'autèntica distribució d'aquests en l'espai original (plafó dret de la figura 25).

Figura 25. MDS exemple 3.19



L'exemple 3.21 descriu una aplicació de la tècnica MDS a un conjunt de punts format per tres subconjunts en els quals les observacions tenen una correlació interna. Cadascun dels grups presenta diferents valors de correlació. En aquest cas, la distància que resulta idònia és la distància de Pearson.

Codi 3.21: Aplicació d'MDS a un conjunt de dades que presenten correlacions internes organitzades en tres grups

```

1  from numpy import *
2  from scaledown import *
3  import pylab as py
4
5  # Matriu de dades (N variables x M observacions):
6  N = 10
7  M = 30
8
9  # Generar dades a partir de 3 grups amb correlacions diferents:
10
11 Xinit1 = random.randn(1,N) # clúster 1 (correlació baixa)
12 X1 = Xinit1
13 for i in range(1,M/3,1):
14     X1 = concatenate((X1,100*Xinit1 + random.rand(1,N)))
15
16 Xinit2 = random.randn(1,N) # clúster 2 (correlació mitjana)
17 X2 = Xinit2
18 for i in range(1,M/3,1):
19     X2 = concatenate((X2,50*Xinit2 + random.rand(1,N)))
20
21 Xinit3 = random.randn(1,N) # clúster 3 (correlació alta)
22 X3 = Xinit3
23 for i in range(1,M/3,1):
24     X3 = concatenate((X3,10*Xinit3 + random.rand(1,N)))

```

```

25
26 #Representació correlacions de cada grup:
27 fig1 = py.figure()
28 py.scatter(X1[0],X1[1],marker='o',hold='on')
29 py.scatter(X2[0],X2[1],marker='x',hold='on')
30 py.scatter(X3[0],X3[1],marker='v',hold='on')
31
32 A = concatenate((X1,X2,X3))
33
34 # Escollir mètrica d'espai original (Euclides o Pearson):
35 metrica = euclidean
36 # mètrica = pearson
37
38 # Tècnica MDS: Retorna la posició de les dades
39 # originals en un espai 2D:
40 mds = scaledown(A,metrica)
41
42 # Representació de les dades en l'espai 2D MDS:
43 fig1 = py.figure()
44 for i in range(0,M/3,1):
45     py.scatter(mds[i][0],mds[i][1],marker='o',c='r')
46 for i in range(M/3,2*M/3,1):
47     py.scatter(mds[i][0],mds[i][1],marker='x',c='g')
48 for i in range(2*M/3,M,1):
49     py.scatter(mds[i][0],mds[i][1],marker='v',c='b')
50 fig1.suptitle('Metrica_Euclidea')
51 py.show()

```

Les figures 26 i 27 reflecteixen la distribució dels punts en l'espai MDS bidimensional quan s'utilitza una mètrica euclidiana (figura 26) i una mètrica de Pearson (figura 27). En aquest cas resulta evident que la mètrica de Pearson permet agrupar els tres subconjunts d'observacions en tres grups ben diferenciats en l'espai de dimensió reduïda.

Figura 26. Projecció MDS 2D de les dades de l'exemple 3.21 utilitzant una mètrica euclidiana

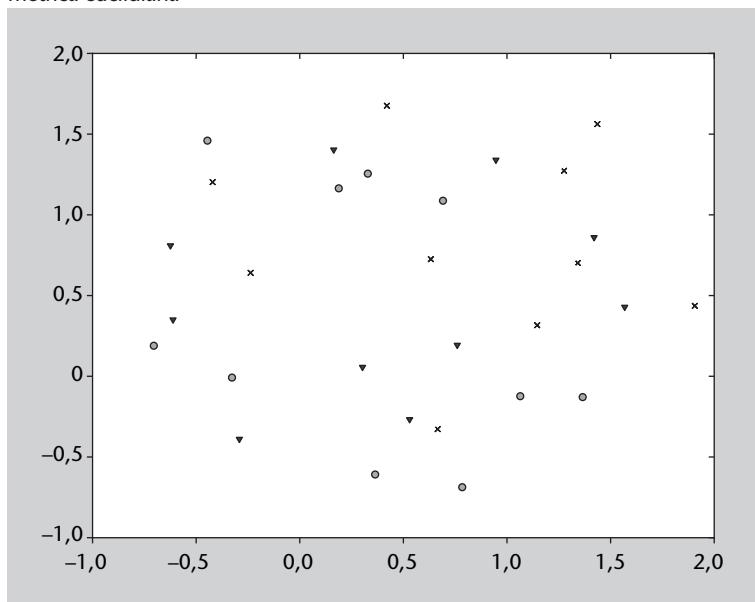
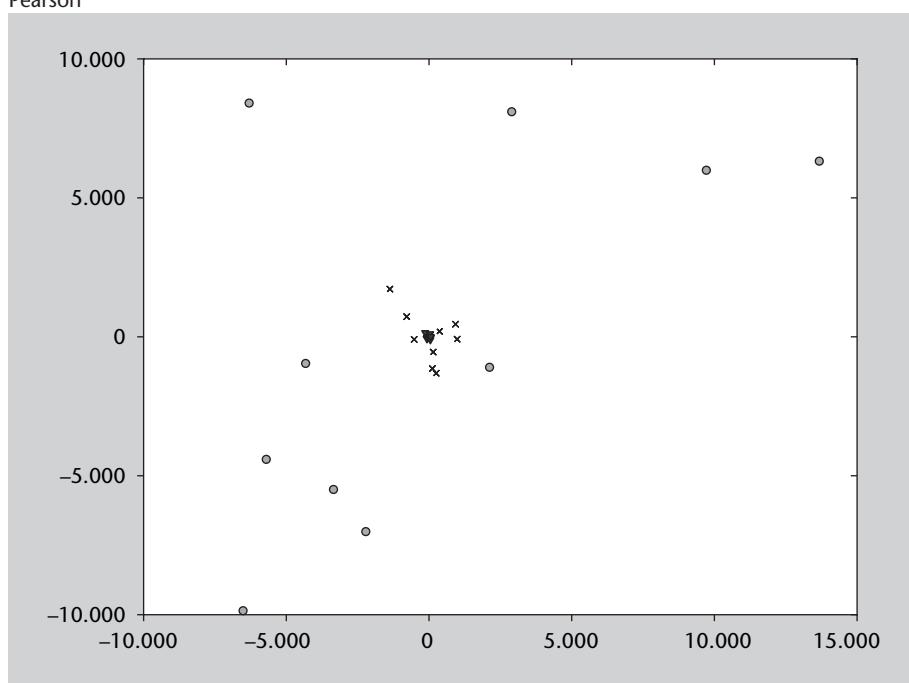


Figura 27. Projecció MDS 2D de les dades de l'exemple 3.21 utilitzant una mètrica de Pearson



4. Classificació

4.1. Introducció

La classificació és una de les tasques de reconeixement de patrons en la qual volem etiquetar un individu a partir de certes propietats que el caracteritzen, entenent com a individu una entitat de qualsevol tipus. A continuació es mostren tres exemples de classificació que utilitzarem al llarg d'aquest apartat:

- Classificació de bolets (Font: problema “mushroom” del repositori UCI (Frank i Asunción, 2010)). Suposem que volem fer una aplicació per a telèfons mòbils que ajudi els aficionats a la recol·lecció de bolets a destriar els bolets verinosos dels comestibles a partir de les seves propietats: forma i color del barret i gruix i color del tronc.
- Classificació de flors (Font: problema “iris” del repositori UCI (Frank i Asunción, 2010)). 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 sobre 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.
- Classificació de documents. Suposem que volem fer una aplicació que, donats els textos que van arribant a una agència de notícies, els classifiqui automàticament per a distribuir-los automàticament als clients interessats. Els temes poden ser: esport, societat, política... o més específicament: futbol, motociclisme, atletisme... Els clients poden ser cadenes de televisió, ràdios, periòdics, revistes de tota índole...

Vegeu també

El problema de la classificació de documents, juntament amb les seves alternatives, es descriu amb més profunditat en el subapartat 4.1.1.

La classificació, des del punt de vista del reconeixement de patrons, involucra tres fases importants: la definició de les classes, la representació de la informació en forma d'atributs i l'aprenentatge mitjançant algorismes.

La definició de classes està determinada en molts problemes. En l'exemple de la classificació de bolets tenim que les classes són: verinós i comestible. En

els altres dos exemples hem de tenir en compte la granularitat de les classes. En la classificació de flors podem escollir entre famílies de flors: rosa, clavell, margarida... O dins d'una família, la flor específica: iris versicolor, iris setosa i iris virgínica. Aquest últim serà el que utilitzarem durant tot l'apartat. En la classificació de documents també ens afecta la granularitat. En aquest cas, a més, ens pot ocórrer que un document tingui més d'una etiqueta: una notícia sobre la relació sentimental entre un futbolista i una cantant pot ser d'esport i societat... La definició de classes té una relació directa amb l'aplicació final de la classificació.

La representació de la informació té a veure amb com representem les característiques dels individus o exemples per etiquetar. Té una relació directa amb el comportament general del classificador que construïm; és tan o més important com l'algorisme d'aprenentatge que utilitzem. Aquest tema s'explica amb més detall en l'apartat 3.

Aquest apartat està dedicat a la tercera de les fases, els algorismes d'aprenentatge per a classificació. Comencem per desenvolupar una mica més el problema de la classificació de documents (o categorització de textos) i per formalitzar el problema. Una vegada fet això donem una tipologia d'aquest tipus d'algorismes i entrem a descriure'l's. Per finalitzar es donen els protocols i mesures d'avaluació.

4.1.1. Categorització de textos

La categorització de textos és un dels problemes que s'intenta resoldre en el camp del processament del llenguatge natural*. Aquest problema consisteix a classificar documents amb una o diverses classes.

* En anglès, *text categorisation* i *natural language processing*.

Un exemple pràctic d'aquest problema, al qual s'estan dedicant molts recursos i esforços, és el tractament de textos per part de les agències de notícies. A aquestes empreses els solen arribar diverses notícies per segon i han d'etiquetar aquestes notícies en funció de la temàtica: esports, política, política internacional, societat... Una de les característiques d'aquest problema és que és multietiqueta*, és a dir, un exemple pot estar associat a més d'una etiqueta.

* En anglès, *multilabel*.

Per a representar els documents en forma d'atributs se solen utilitzar conjunts de paraules*. Aquesta tècnica ve del camp de la recuperació de la informació**. Els conjunts de paraules solen aparèixer en la bibliografia de dues maneres: com a conjunts de paraules pròpiament dits (sobretot quan es treballa a escala de frase) o com a conjunts de paraules anotant el nombre de vegades que apareixen en el document (o la seva freqüència d'aparició).

* En anglès, *bag of words* o *vector space model (VSM)*.

** En anglès, *information retrieval*.

Quan es treballa amb conjunts d'atributs, se solen processar les dades per a eliminar les paraules sense contingut semàntic (com preposicions, articles...),

per a eliminar signes de puntuació i per a treballar directament amb els lemes de les paraules (s'elimina el gènere, nombre i temps verbal). Per a poder fer això, s'utilitzen dos tipus de recursos: llistes de paraules sense contingut semàntic* i processadors lingüístics.

* En anglès, *lists of stop words*.

Enllaç d'interès

A la pàgina web del professor Lluís Padró (<http://www.lsi.upc.edu/~padro>) es poden trobar llistes de paraules sense contingut semàntic per a l'anglès, el castellà i el català. Les teniu disponibles en el repositori de l'assignatura.

El segon tipus de recurs són els processadors lingüístics. En solen incloure tockenitzadors, lematitzadors, etiquetadors morfosintàctics, analitzadors sintàctics superficials... per a diferents llengües. Per a aquest problema, amb el tockenitzador i el lematitzador ja complirem amb les nostres necessitats. Hi ha diversos processadors lingüístics lliures disponibles a la xarxa. Dos són el FreeLing* i l'NLTK**.

* <http://nlp.lsi.upc.es/freeling>

** <http://www.nltk.org>

Un cas particular de la categorització de textos són els filtres antibrossa*. En aquest problema cal pensar com hem d'afegir com a atributs els components dels correus, i també les adreces d'origen o l'assumpte.

* En anglès, *anti-spam*.

El conjunt de dades més utilitzat en categorització de textos és el Reuters-21578*. Els conjunts de dades de textos es denominen *corpus*. Aquest corpus ha estat generat a partir de dades de l'agència de notícies Reuters. Els documents que conté estan sense processar i tenen associada una sèrie de classes o categories. Aquest conjunt de dades també està disponible en el repositori de la UCI (Frank i Asunción, 2010).

* <http://www.daviddewLewis.com/resources/testcollections/reuters21578/>

El corpus* següent és un subconjunt de l'anterior. T. Joachims va processar una part del corpus anterior i va calcular les freqüències d'aparició de les diferents paraules en el document. Aquesta manera de representar la informació es denomina VSM**. La tasca associada a aquest corpus és distingir si els documents pertanyen o no a la classe *adquisicions corporatives*. El corpus està pensat per a destriar si un document és d'aquesta classe o no. El conjunt de dades conté 1.000 exemples positius i 1.000 de negatius en el conjunt d'entrenament i 300 de cada en el conjunt de test. El nombre d'atributs ascendeix a 9.947 i contenen informació de la freqüència relativa de les paraules en els documents.

* <http://svmlight.joachims.org>

** De l'anglès, *vector space model*. També sol rebre el nom de *bossa de paraules* (de l'anglès *bag of words*).

El tercer conjunt de dades és el TechTC-100*, que ha estat recopilat per Gabrilovich i Markovitch. Aquest conjunt consta d'una sèrie de pàgines web que estan associades a una ontologia o jerarquia. Les dades estan disponibles sense processar i processades. Aquest procés consisteix a generar el VSM per als documents desant el nombre de vegades que apareix un terme o paraula en cada document. Les classes són les branques de la jerarquia ODP** a la qual pertanyen les pàgines.

* <http://techtc.cs.technion.ac.il/techtc100/techtc100.html>

** <http://www.dmoz.org>

4.1.2. Aprendentatge automàtic per a classificació

En aquest subapartat veurem la formalització del problema de classificació tal com s'usa normalment en la bibliografia del camp de l'aprendentatge automàtic i el reconeixement de patrons.

L'objectiu de l'aprendentatge automàtic* per a classificació (o aprenentatge supervisat) consisteix a induir una aproximació (model o hipòtesi) h d'una funció desconeguda f definida des d'un espai d'entrada X cap a un espai discret i desordenat $Y = 1, \dots, K$, donat un conjunt d'entrenament S .

* En anglès, *machine learning*.

Regressió

Quan l'espai de sortida és continu, el problema per tractar serà de *regressió* en lloc de classificació.

El conjunt d'entrenament $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$ conté n exemples d'entrenament, que corresponen a parells (x, y) en què $x \in X$ i $y = f(x)$. El component x de cada exemple és un vector $x = (x_1, \dots, x_n)$ d'atributs amb valors continus o discrets que descriuen la informació rellevant o propietats de l'exemple. Els valors de l'espai de sortida Y associats a cada exemple són les classes del problema. Així, cada exemple d'entrenament queda totalment caracteritzat per un conjunt de parells atribut-valor i una etiqueta de classe.

Clustering

Quan no utilitzem les etiquetes de classe y_i en el conjunt d'entrenament $S = \{x_1, \dots, x_n\}$ parlem de *clustering* o aprendentatge no supervisat. Aquest tema es descriu en el subapartat 2.3. d'aquest mòdul.

Donat un conjunt d'entrenament S , un algorisme d'entrenament induceix un classificador h que correspon a una hipòtesi sobre la funció f . En fer això, l'algorisme pot escollir entre un conjunt de possibles funcions H anomenat *espai d'hipòtesis*. Els diferents algorismes d'aprendentatge difereixen en l'espai d'hipòtesis que tenen en compte (alguns tractaran funcions lineals, uns altres crearan hiperplans que particionen dominis...); en el llenguatge de representació que utilitzen (per exemple: arbres de decisió, conjunts de probabilitats condicionals, xarxes neuronals...) i en el *biaix* que usen en escollir la “míller” hipòtesi entre les que són compatibles amb el conjunt entrenament (per exemple: simplicitat, marge màxim...).

Partint de nous vectors x , utilitzarem h per a predir els valors corresponents de y per a classificar els nous exemples, tot esperant que coincideixin amb f en la majoria de casos, o el que és equivalent, que es produeix el mínim nombre d'errors.

La mesura de l'error dels nous exemples (o exemples no vistos) es diu *error de generalització**. Resulta obvi que l'error de generalització no es pot minimitzar sense conèixer *a priori* la funció f o la distribució $P(X, Y)$. Això fa que necessitem un principi d'inducció. La manera més usual de procedir consisteix a intentar minimitzar l'error sobre el conjunt d'entrenament (o error empíric). Aquest principi es coneix com a *minimització empírica del risc*** i dóna una bona estimació de l'error de generalització en presència de molts exemples d'entrenament.

Lectures complementàries

S. Kulkarni; G. Harman (2011). *An Elementary Introduction to Statistical Learning Theory*. EUA: John Wiley and Sons, Inc.

* En anglès, *generalisation or true error*.

** En anglès, *empirical risk minimisation*.

No obstant això, en dominis amb pocs exemples d'entrenament, forçar que la minimització de l'error empíric tendeixi a zero pot fer que se sobreentreni* el conjunt d'entrenament i que es generalitzi de manera errònia. La complexitat de les funcions induïdes** i la presència de soroll i exemples atípics*** (mal classificats o inconsistents) en el conjunt d'entrenament també pot augmentar el risc de sobreentrenament. En definir un marc experimental és necessari establir un compromís entre l'error experimental i la complexitat dels classificadors induïts per a garantir un error de generalització petit.

* En anglès, *overfitting*.

** La complexitat del model es mesura amb l'anomenada dimensió de Vapnik-Chervonenkis.

*** En anglès, *outliers*.

Tota la nomenclatura utilitzada en els dos paràgrafs anteriors ve de la *teoria de l'aprenentatge estadístic**. La manera clàssica de tractar aquest tema és a partir del *biaix*** i la variància. El biaix d'un estimador correspon a la diferència entre la seva esperança matemàtica i el valor que està estimant. Tot el que res tringeix l'espai de cerca està relacionat amb el biaix. La complexitat del model es mesura per mitjà de la variància, que mesura la dispersió dels valors entorn de l'esperança. La complexitat del model està directament relacionada amb la variància i inversament relacionada amb el biaix. L'error de generalització està format per la suma del biaix i la variància del model. En la construcció de models sempre hem de tractar amb un compromís entre el biaix i la variància.

* En anglès, *statistical learning theory*.

** En anglès, *bias*.

A tall d'exemple, la taula 7 mostra les característiques de dos exemples del conjunt de dades dels bolets descrit anteriorment. Els atributs del primer exemple $x = (x_1, x_2, x_3, x_4)$ són *cap-shape*, *cap-color*, *gill-size* i *gill-color* amb valors *convex*, *brown*, *narrow* i *black*, respectivament; el valor de y és *poisonous*. El conjunt Y conté les etiquetes *{poisonous, edible}*.

Taula 7. Exemple del tractament de bolets

class	cap-shape	cap-color	gill-size	gill-color
poisonous	convex	brown	narrow	black
edible	convex	yellow	broad	black

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

Tota la formulació que hem vist fins aquí correspon a problemes monoetiqueta, en els quals cada exemple només té associada una única etiqueta. Per a formalitzar els problemes multietiqueta, com la categorització de textos, hem de canviar les característiques de la funció $f(x)$. En els problemes monoetiqueta aquesta funció retorna un únic valor de $Y = \{1, \dots, k\}$, en què k és el nombre de classes; en els problemes multietiqueta, la funció $f(x)$ retorna un conjunt de valors de Y , que pot ser buit.

4.1.3. Tipologia d'algorismes per a classificació

Podem classificar els algorismes en funció del principi d'inducció que usen per a adquirir els models o regles de classificació a partir dels exemples. Aquests mètodes estan basats en: probabilitats, distàncies, regles o *kernels*. Aquesta tipologia no té la intenció de ser exhaustiva o única. Per descomptat, la combi-

nació de diferents paradigmes és una altra possibilitat que s'està utilitzant en els últims temps.

Els quatre subapartats següents estan enfocats a explicar aquests paradigmes i alguns dels seus algorismes més característics. L'últim subapartat ens presenta els protocols d'avaluació que s'utilitzen en els processos de classificació.

4.2. Mètodes basats en models probabilístics

Els mètodes estadístics soLEN estimar un conjunt de paràmetres probabilístics, que expressen la probabilitat condicionada de cada classe donades les propietats d'un exemple (descrit en forma d'atributs). A partir d'això, aquests paràmetres podEN ser combinats per a assignar les classes que maximitzen les seves probabilitats a nous exemples.

4.2.1. Naïve Bayes

Suposem que volem fer una aplicació per a telèfons mòbils que ajudi els afeccionats a la recol·lecció de bolets a destriar els bolets verinosos dels comestibles a partir de les seves propietats: forma i color del barret i gruix i color del tronc. La taula 8 mostra exemples d'aquest tipus de dades.

Taula 8. Conjunt d'entrenament

class	cap-shape	cap-color	gill-size	gill-color
poisonous	convex	brown	narrow	black
edible	convex	yellow	broad	black
edible	bell	white	broad	brown
poisonous	convex	white	narrow	brown
edible	convex	yellow	broad	brown
edible	bell	white	broad	brown
poisonous	convex	white	narrow	pink

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

El Naïve Bayes és el representant més simple dels algorismes basats en probabilitats. Està basat en el teorema de Bayes.

L'algorisme de Naïve Bayes el van descriure Duda i Hart en la seva forma més clàssica el 1973.

L'algorisme de Naïve Bayes classifica nous exemples $x = (x_1, \dots, x_m)$ assignant-los la classe k que maximitza la probabilitat condicional de la classe donada la seqüència observada d'atributs de l'exemple. És a dir,

$$\operatorname{argmax}_k P(k|x_1, \dots, x_m) = \operatorname{argmax}_k \frac{P(x_1, \dots, x_m|k)P(k)}{P(x_1, \dots, x_m)} \approx \operatorname{argmax}_k P(k) \prod_{i=1}^m P(x_i|k)$$

en què $P(k)$ i $P(x_i|k)$ s'estimen a partir del conjunt d'entrenament, utilitzant les freqüències relatives (estimació de la màxima versemblança*).

* En anglès, *maximum likelihood estimation*.

Exemple d'aplicació

La taula 8 mostra un exemple de conjunt d'entrenament en el qual hem de detectar si un bolet és verinós o no en funció de les seves propietats.

Durant el procés d'entrenament començarem per calcular $P(k)$ per a cadascuna de les classes $k \in Y$. Així, aplicant una estimació de la màxima versemblança obtenim $P(poisonous) = 3/7 = 0,43$ i $P(edible) = 4/7 = 0,57$. El segon pas consisteix a calcular $P(x_i|k)$ per a cada parella atribut-valor i per a cada classe, de la mateixa manera que en el cas anterior. La taula 9 mostra els resultats d'aquest procés. L'1 de la cel·la corresponent a la fila *cap-shape: convex* i columna *poisonous* surt del fet que els tres exemples etiquetats com a *poisonous* tenen el valor *convex* per a l'atribut *cap-shape*.

Taula 9. Valors de $P(x_i|k)$

atribut-valor	poisonous	edible
cap-shape: convex	1	0,5
cap-shape: bell	0	0,5
cap-color: brown	0,33	0
cap-color: yellow	0	0,5
cap-color: white	0,67	0,5
gill-size: narrow	1	0
gill-size: broad	0	1
gill-color: black	0,33	0,25
gill-color: brown	0,33	0,75
gill-color: pink	0,33	0

Amb això haurem acabat el procés d'entrenament. A partir d'aquí, si ens arriba un exemple nou com el que mostra la taula 10 haurem d'aplicar la fórmula $\text{argmax}_k P(k) \prod_{i=1}^m P(x_i|k)$ per a classificar-lo.

Taula 10. Exemple de test

class	cap-shape	cap-color	gill-size	gill-color
poisonous	convex	brown	narrow	black

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

Comencem per calcular la part del valor *poisonous*. Per a això haurem de multiplicar entre si les probabilitats: $P(poisonous)$, $P(cap - shape - convex|poisonous)$, $P(cap - color - brown|poisonous)$, $P(gill - size - narrow|poisonous)$ i $P(gill - color - black|poisonous)$; això equival a un valor de 0,05. Fent el mateix procés per a la classe *edible* obtenim un valor de 0. Per tant, la classe que maximitza la fórmula de les probabilitats és *poisonous*, amb la qual cosa el mètode està classificant correctament l'exemple de test, donat aquest conjunt d'entrenament.

Anàlisi del mètode

Un dels problemes del Naïve Bayes, que ha estat esmentat freqüentment en la bibliografia, és que l'algorisme assumeix la independència dels diferents atributs que representen un exemple. 

A tall d'exemple, és poc probable que el color del tronc d'un bolet no estigui relacionat amb el color del barret.

Fins ara hem parlat exclusivament d'atributs nominals. En el cas de tenir un problema representat amb algun atribut continu, com els numèrics, és necessari algun tipus de procés. Hi ha diverses maneres de fer-ho; una consisteix a categoritzar-los, dividint el continu en intervals. Una altra opció consisteix a assumir que els valors de cada classe segueixen una distribució gaussiana i aplicar la fórmula:

$$P(x = v|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} \exp\left(-\frac{(v - \mu_c)^2}{2\sigma_c^2}\right)$$

en què x correspon a l'atribut, v al seu valor, c a la classe, μ_c a la mitjana de valors de la classe c i σ_c^2 a la seva desviació estàndard.

L'algorisme presenta un problema numèric que cal tenir en compte per a certs conjunts de dades. Quan ens trobem amb un exemple de test que té alguna parella atribut-valor que no ha aparegut en el conjunt d'entrenament, no tindrem cap valor per a $P(x_i|k)$. La taula 11 mostra un exemple per al conjunt d'entrenament anterior; el valor “white” de l'atribut “gill-color” no apareix en el conjunt d'entrenament. Per a solucionar aquest problema se sol aplicar alguna tècnica de *suavitzat**.

* En anglès, *smoothing*.

Taula 11. Exemple de test (suavitzat)

class	cap-shape	cap-color	gill-size	gill-color
edible	bell	yellow	broad	white

Font: problema “mushroom” del repositori UCI (Frank i Asunción, 2010).

Un exemple de tècnica de suavitzat per a aquest algorisme consisteix a substituir la $P(x_i|k)$ que conté el comptador a zero per $P(k)/n$, en què n correspon al nombre d'exemples d'entrenament.

Una altra característica per destacar és que quan el conjunt d'entrenament no està balancejat, tendeix a classificar els exemples envers la classe que té més exemples dins del conjunt d'entrenament.

Conjunt d'entrenament equilibrat

Un conjunt d'entrenament està equilibrat quan té el mateix nombre d'exemples de cada una de les classes que conté.

Dos dels avantatges que presenta el mètode són la simplicitat i l'eficiència computacional.

No obstant això, i malgrat els inconvenients, aquest mètode ha estat molt utilitzat històricament i s'han obtingut bons resultats per a molts conjunts de dades. Això ocorre quan el conjunt d'entrenament representa bé les distribucions de probabilitat del problema.

Lectura complementària

Aquesta tècnica de suavitzat apareix a: H. T. Ng (1997). “Exemplar-based Word Sense Disambiguation: Some Recent Improvements”. En les actes de la conferència *Empirical Methods in Natural Language Processing, EMNLP*.

Implementació en Python

En intel·ligència artificial s'ha usat clàssicament la programació funcional (representada pel llenguatge LISP) i la programació lògica (representada per PROLOG). En aquest apartat els programes es donaran en Python, però usant la part de programació funcional que conté. El programa 4.1 correspon a la implementació, seguint aquestes directrius, de l'algorisme Naïve Bayes.

Codi 4.1: Naïve Bayes en Python

```

1  from functools import reduce
2
3  # declaració de funcions
4  def contar(l):
5      p = {}
6      for x in l:
7          p.setdefault(x, 0)
8          p[x] += 1
9      return p
10
11 def comptar2(l, k):
12     p = {}
13     for i in range(len(l)):
14         p.setdefault(k[i], {})
15         p[k[i]].setdefault(l[i], 0)
16         p[k[i]][l[i]] += 1
17     return p
18
19 def Pxik(atr, N, k):
20     if atr in N[k]:
21         return N[k][atr] / Nk[k]
22     else:
23         return Nk[k] / n ** 2
24
25 def classify(t):
26     l = [(k, Nk[k] / n *
27           reduce(lambda x, y: x * y,
28                 map(Pxik, t, Nxik,
29                     [k for a in range(len(t))])))
30         for k in Nk.keys()]
31     return max(l, key=lambda x: x[1])[0]
32
33 # càrrega de l'arxiu
34 l = list(map(lambda l: (l.strip()).split(','),
35             open('mushroom.data.txt', 'r').readlines()))
36
37 # construcció del training: 2 de cada 3
38 train = list(map(lambda x: x[1],
39                  filter(lambda v: v[0] % 3 != 0, enumerate(l))))
40 n = len(train)
41
42 # construcció del test: 1 de cada 3
43 test = list(map(lambda x: x[1],
44                  filter(lambda v: v[0] % 3 == 0, enumerate(l))))
45
46 # transposada del training
47 m = list(zip(*train))
48
49 # Numerador de  $P(k)$ 
50 Nk = contar(m[0])
51
52 # Numerador de  $P(xi|k)$ 
53 Nxik = [comptar2(m[1], m[0]) for i in range(1, len(m))]
54
55 # Classificació
56 classes = list(map(lambda x: x.pop(0), test))
57 prediccions = list(map(classify, test))

```

```

58
59 # Nombre de correctes
60 print('Prec.: ', len(list(filter(lambda x: x[0] == x[1],
61                               zip(*[prediccions, classes])))))
62     / len(test) * 100, '%')

```

La implementació preveu l'ús de la tècnica de suavitzat comentada en el sub-apartat anterior. Aquest programa servirà per a qualsevol conjunt de dades en què tots els seus atributs siguin nominals (sense que importi el nombre), sense valors absents i que tingui la classe com a primera columna. Estan marcats, mitjançant comentaris, els diferents processos que fa.

Si apliquem el programa a l'arxiu de dades del problema “mushroom” de la UCI (Frank i Asunción, 2010) obtindrem una precisió del 55,7%. Aquest programa selecciona com a conjunt de test un de cada tres exemples de l'arxiu. La resta d'exemples pertanyerà al conjunt d'entrenament.

Vegeu també

Els arxius de dades i el programa estan disponibles en el repositori de l'assignatura.

4.2.2. Màxima entropia

Suposem que volem fer una aplicació per a telèfons mòbils que ajudi els afecionats a la recol·lecció de bolets a destriar els bolets verinosos dels comestibles a partir de les seves propietats: forma i color del barret. La taula 12 mostra exemples d'aquest tipus de dades.

Taula 12. Conjunt d'entrenament

class	cap-shape	cap-color
poisonous	convex	brown
edible	convex	yellow
edible	bell	white
poisonous	convex	white

Font: problema “mushroom” del repositori UCI (Frank i Asunción, 2010).

L'algorisme de màxima entropia proporciona una manera flexible de combinar evidències estadístiques de diferents fonts. L'estimació de probabilitats no assumeix coneixements previs de les dades i s'ha provat que és molt robust.

El principi de màxima entropia s'ha utilitzat en molts àmbits, i no és un algorisme de classificació. En aquest subapartat anem a veure l'adaptació del principi de la màxima entropia per a crear un classificador.

Donat un conjunt d'entrenament $\{(x_1, y_1), \dots, (x_n, y_n)\}$ en què n és el nombre d'exemples d'entrenament, la nostra tasca serà trobar les probabilitats condicionades $p(y|f)$, en què f són els atributs que ens permeten definir el model de classificació.

Algorisme de màxima entropia

L'algorisme de màxima entropia va aparèixer per primera vegada el 1957 en dos articles d'E. T. Jaynes.

Lectura complementària

En aquest subapartat utilitzarem l'aproximació que es fa del principi de màxima entropia a: A. L. Berger, S. A. Della Pietra; V. J. Della Pietra (1996). “A Maximum Entropy Approach to Natural Language Processing”. *Computational Linguistics* (vol. 1, núm. 22).

El principi de màxima entropia consisteix a definir un problema a partir de restriccions, i a partir d'això definim la distribució de probabilitats que sigui més uniforme. Una mesura matemàtica de la uniformitat d'una distribució condicional $p(y|x)$ ens la dóna l'entropia condicional:

$$H(p) = - \sum_{x,y} \tilde{p}(x)p(y|x) \log(p(y|x))$$

en què $\tilde{p}(x)$ correspon a la distribució de probabilitat empírica.

Representarem les parelles atribut-valor a partir de funcions $f_i(x,y)$ que assignaran un valor d'1 si l'exemple x correspon com a valor de l'atribut a la parella representada per f_i , o 0 en cas contrari. A manera d'exemple, si x_1 correspon al primer exemple de la taula 12 i f_1 a l'atribut corresponent a la parella (*cap-shape*, *convex*), la funció quedaria com:

$$f_1(x,y) = \begin{cases} 1 & \text{si } x \text{ té } \textit{convex} \text{ coma valor de l'atribut } \textit{cap-shape} \\ 0 & \text{si no el té} \end{cases}$$

i el valor de $f_1(x_1,y_1)$ seria 1.

El valor esperat de f pel que fa a la distribució empírica $\tilde{p}(x,y)$ és:

$$\tilde{p}(f) = \sum_{x,y} \tilde{p}(x,y)f(x,y)$$

i el valor esperat de f pel que fa al model $p(y|x)$ que volem:

$$p(f) = \sum_{x,y} \tilde{p}(x)p(y|x)f(x,y)$$

que ens genera la restricció:

$$p(f) = \tilde{p}(f)$$

o

$$\sum_{x,y} \tilde{p}(x,y)f(x,y) = \sum_{x,y} \tilde{p}(x)p(y|x)f(x,y)$$

Si tenim el conjunt de restriccions $C = \{p \in P | p(f_i) = \tilde{p}(f_i), \forall 1 \leq i \leq n\}$, el principi de màxima entropia tracta de buscar la distribució de probabilitat amb entropia màxima:

$$p^* = \operatorname{argmax}_{p \in C} H(p)$$

Afegirem les restriccions que segueixen per a garantir que p són distribucions de probabilitat condicional:

$$p(y|x) \geq 0, \forall x, y$$

$$\sum_y p(y|x) = 1, \forall x$$

Per a atacar aquest problema, aplicarem el mètode dels multiplicadors de Lagrange de la teoria de l'optimització de restriccions:

$$\begin{aligned} \zeta(p, \Lambda, \gamma) = & -\sum_{x,y} \tilde{p}(x)p(y|x) \log(p(y|x)) + \\ & + \sum_i \lambda_i (\sum_{x,y} \tilde{p}(x,y)f_i(x,y) - \tilde{p}(x)p(y|x)f_i(x,y)) + \\ & + \gamma \sum_x p(y|x) - 1 \end{aligned}$$

Vegeu també

El mètode dels multiplicadors de Lagrange s'estudia en el subapartat 5.2. d'aquest mòdul.

De les solucions a aquest problema tindrem:

$$p_\lambda(y|x) = \frac{1}{Z_\lambda(x)} \exp \left(\sum_i \lambda_i f_i(x,y) \right)$$

en què $Z_\lambda(x)$ correspon a un factor de normalització:

$$Z_\lambda(x) = \sum_i \exp \left(\sum_i \lambda_i f_i(x,y) \right)$$

Ús en Python

L'algorisme de la màxima entropia està implementat en la biblioteca *SciPy**. Aquesta biblioteca porta com a exemples d'ús de la màxima entropia els exemples que apareixen en l'article de Berger i altres.

* <http://www.scipy.org>

Referència bibliogràfica

A. L. Berger; S. A. Della Pietra; V. J. Della Pietra (1996). "A Maximum Entropy Approach to Natural Language Processing". *Computational Linguistics* (vol. 1, núm. 22).

4.3. Mètodes basats en distàncies

Els mètodes d'aquesta família classifiquen els nous exemples a partir de mesures de similitud o distància. Això es pot fer comparant els nous exemples amb conjunts (un per classe) de prototips i assignant la classe del prototip més proper, o buscant en una base d'exemples quin és el més proper.

4.3.1. kNN

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 sobre 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 13 mostra exemples d'aquest tipus de dades.

Taula 13. 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
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).

Un dels algorismes representatius d'aquesta família és el kNN (k veïns més propers*). En aquest algorisme la classificació de 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 de nous exemples**.

* En anglès, *k nearest neighbours*.

** Per aquesta raó de vegades és anomenat *aprenentatge mandrós* (en anglès, *lazy learning*).

Una part molt important d'aquest mètode és la definició de la mesura de distància (o similitud) apropiada per al problema per tractar. Aquesta hauria de tenir en compte la importància relativa de cada atribut i ser eficient computacionalment. El tipus de combinació per a escollir el resultat entre els k exemples més propers i el valor de k mateixa també són qüestions per decidir entre diverses alternatives.

Aquest algorisme, en la seva forma més simple, desa en memòria tots els exemples durant el procés d'entrenament i la classificació de 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 per classificar $x = (x_1, \dots, x_m)$ i tots els exemples desats $x_i = (x_{i1}, \dots, x_{im})$. Una de les distàncies més utilitzades és l'euclidiana:

* Per aquesta raó es coneix també com basat en memòria, en exemples, en instàncies o en casos.

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

Exemple d'aplicació

La taula 13 mostra un conjunt d'entrenament en el qual 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 14, hem de calcular les distàncies entre el nou exemple i tots els del conjunt d'entrenament. La taula 15 mostra aquestes distàncies.

Taula 14. 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 15. 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 distàncies respectives: 0,5, 0,2 i 2,5. Les seves classes corresponen a *setosa*, *setosa* i *versicolor*. En aquest cas l'assignarem també a *setosa* per ser la classe més freqüent. En tots dos casos el resultat és correcte.

Anàlisi del mètode

Un aspecte per tenir en compte té a veure amb l'eficiència computacional; l'algorisme fa tots els càlculs en el procés de classificació. Així, encara que és un mètode ràpid globalment, hem de tenir en compte que el procés de classificació no ho és. Això pot arribar a ser crític per a aplicacions de temps real que necessitin una resposta ràpida.

En el moment en què vulguem aplicar aquest algorisme a un problema, la primera decisió que hem de prendre és la mesura de distància, i la segona el valor de k . Se sol escollir un nombre imparell o primer per minimitzar la possibilitat d'empats en les votacions. La qüestió següent per decidir és el tractament dels empats quan la k és superior a 1. Alguns heurístics possibles són: no donar prediccio en cas d'empat, donar la classe més freqüent en el conjunt d'entrenament entre les classes seleccionades per a votar... Se sol escollir l'heurístic en funció del problema per tractar.

Les distàncies més utilitzades són l'euclidiana i la de Hamming, en funció del tipus d'atributs que tinguem. La primera s'utilitza majoritàriament per a atributs numèrics i la segona per a atributs nominals o binaris. Altres tipus de distàncies poden ser utilitzades dependent del problema per tractar. Un exem-

Distància de Hamming

La distància de Hamming és $dh(x, x_i) = \sum_{j=1}^m \delta(x_j, x_{ij})$ en què $\delta(x_j, x_{ij})$ és la distància entre dos valors que correspon a 0 si $x_j = x_{ij}$ i a 1 si són diferents.

ple d'això és la distància MVDM. Aquesta mesura substitueix la d'Hamming en el tractament d'atributs nominals i ha donat bons resultats en problemes de processament del llenguatge natural semblants a la categorització de textos. L'inconvenient principal és l'alt cost computacional.

Una variant de l'algorisme molt utilitzada és la utilització d'exemples ponderats. Consisteix en la introducció d'una modificació en l'esquema de votació dels k veïns més propers, que fa la contribució de cada exemple proporcional a la seva importància. Aquesta importància es mesura tenint en compte la proximitat a l'exemple de test.

La ponderació dels atributs també s'ha utilitzat. Això consisteix a fer un "rànquing" dels atributs en funció de la seva rellevància i fer que la seva contribució afecti el càlcul de la distància. Els pesos dels atributs es poden calcular de diverses maneres; una és la distància RLM*. A manera d'exemple, aplicar aquest concepte a la distància de Hamming, una vegada calculats els pesos, dóna com a resultat:

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

Una altra manera de tractament d'atributs és la selecció, sobretot en problemes en els quals el nombre és important. La selecció d'atributs consisteix a reduir el nombre d'atributs dels exemples. La idea és doble: selecció dels més rellevants i eliminació dels que produeixen soroll. Una manera de fer això consisteix a dividir el conjunt d'entrenament en dos: un d'entrenament més petit i un altre anomenat de *validació*. S'aplica un algorisme d'optimització (com els descrits en l'apartat 5) escollint com a funció objectiu el resultat de la classificació sobre el conjunt de validació. El conjunt d'entrenament d'aquests classificador s és la part de l'entrenament que no pertany al de validació. Una vegada seleccionats els atributs, s'entrena el classificador final amb tot el conjunt d'entrenament i es prova amb el de test real. Una altra manera de fer això consisteix a aplicar algun dels mètodes descrits en l'apartat 3, com el *PCA*.

Un dels grans avantatges d'aquest mètode és la conservació d'excepcions en el procés de generalització, i un dels grans inconvenients coneguts és la sensibilitat a processos de selecció d'atributs.

Implementació en Python

El programa 4.2 correspon a la implementació en Python del kNN bàsic. En aquesta versió s'ha escollit com a distància l'euclidiana i 3 com a valor de la k . Aquest programa funcionarà per a tots els problemes amb dades numèriques sense valors absents que tinguin la classe com a última columna.

MVDM

MVDM són les sigles de l'anglès *modified value difference metric*. Proposta per: **S. Cost; S. Salzberg** (1993). "A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features". *Machine Learning* (núm. 10)

* **Distància per Ramón López de Mántaras a: R. López de Mántaras** (1991). "A distance-based attribute selection measure for decision tree induction". *Machine Learning* (núm. 6, pàg. 81-92).

Lectura recomanada

En la referència següent s'utilitzen algoritmes genètics per a seleccionar atributs i l'ajust de paràmetres d'un algorisme basat en kNN: **B. Decadt; V. Hoste; W. Daelemans; A. van den Bosch** (2004). "GAMBL, Genetic Algorithm Optimization of Memory-Based WSD". En les actes de l'*International Workshop on Evaluating WSD Systems, Senseval* (núm. 3)

Codi 4.2: kNN en Python

```

1 # parametros
2
3 k = 3
4
5 # declaració de funcions
6
7 def deeuclidiana(x, y):
8     return sum(map(lambda a, b: (float(a) - float(b)) ** 2,
9                   x, y)) ** (1 / 2)
10
11 def comptar(l):
12     p = {}
13     for x in l:
14         p.setdefault(x, 0)
15         p[x] += 1
16     return p
17
18 def classify(t):
19     ds = list(map(deeuclidiana, train, [t for x in range(len(train))]))
20     kcl = comptar([sorted([(ds[i], classesTrain[i]),
21                           for i in range(len(train))]),
22                           key=lambda x: x[0])[i][1]
23                           for i in range(k)])
24     return max([(x, kcl[x]) for x in kcl.keys()],
25                key=lambda x: x[1])[0]
26
27 # carrega de l'arxiu
28 l = list(map(lambda l: (l.strip()).split(','), 
29             open('iris.data.txt', 'r').readlines()))
30
31 # construcció del training: 2 de cada 3
32 train = list(map(lambda x: x[1],
33                  filter(lambda v: v[0] % 3 != 0, enumerate(l))))
34 nc = len(train[0])
35 classesTrain = list(map(lambda x: x.pop(nc - 1), train))
36
37 # construcció del test: 1 de cada 3
38 test = list(map(lambda x: x[1],
39                  filter(lambda v: v[0] % 3 == 0, enumerate(l))))
40 classesTest = list(map(lambda x: x.pop(nc - 1), test))
41
42 # Classificació
43 prediccions = list(map(classify, test))
44
45 # Nombre de correctes
46 print('Prec.: ', len(list(filter(lambda x: x[0] == x[1],
47                               zip(*[prediccions, classesTest])))) /
48     len(test) * 100, '%')

```

Si apliquem el programa a l'arxiu de dades del problema “iris” de la UCI (Frank i Asunción, 2010) obtindrem una precisió del 98%. Aquest programa selecciona com a conjunt de test un de cada tres exemples de l'arxiu. La resta d'exemples pertanyerà al conjunt d'entrenament.

Repositori

Els arxius de dades i el programa estan disponibles en el repositori de l'assignatura.

4.3.2. Classificador lineal basat en distàncies

Una manera molt semblant a la de l'algorisme anterior d'abordar el problema de la classificació de flors es basa en l'ús de centres de massa o centroides. El model de classificació d'aquest algorisme consta d'un centroïde, que representa cadascuna de les classes que apareixen en el conjunt d'entrenament*.

* Aquest mètode també rep el nom de mètode basat en centroides o en centres de masses.

El valor de cada atribut del centroide es calcula com la mitjana del valor del mateix atribut de tots els exemples del conjunt d'entrenament que pertanyen a la seva classe. La fase de classificació consisteix a aplicar l'1NN amb distància euclidiana, seleccionant com a conjunt d'entrenament els centroides calculats prèviament.

Exemple d'aplicació

Aplicarem com a exemple l'algorisme als exemples de la taula 13. El procés d'entrenament consisteix a calcular els centroides. La taula 16 mostra el resultat. A manera d'exemple, el 5 de l'atribut *sepal-length* del centroide *setosa* surt de la mitjana de 4,9 i 5,1 de la taula 13.

Taula 16. Centroides

class	sepal-length	sepal-width	petal-length	petal-width
setosa	5	3,25	1,4	0,2
versicolor	5,85	2,9	4,15	1,35
virginica	6,25	2,75	5,55	1,9

A partir d'això, si ens arriba un exemple nou com el que mostra la taula 14, hem de calcular les distàncies entre el nou exemple i tots els centroides per a l'1NN. La taula 17 mostra aquestes distàncies. En escollir el més proper (distància 0,2) que té per classe *setosa*, veiem que l'exemple queda ben classificat.

Taula 17. Distàncies

0,2	3,1	4,6
-----	-----	-----

Anàlisi del mètode

Aquest algorisme es pot aplicar utilitzant un producte escalar i s'ha utilitzat molt des d'aquest altre punt de vista. En el subapartat 4.5.1. es planteja aquest nou punt de vista i s'analitza en més profunditat.

Implementació en Python

El programa 4.3 correspon a la implementació en Python del classificador lineal. Aquest programa funcionarà per a tots els problemes amb dades numèriques sense valors absents que tinguin la classe com a última columna.

Codi 4.3: Classificador lineal basat en distàncies en Python

```

1 from functools import reduce
2 # declaració de funcions
3
4 def deeuclidiana(x, y):
5     return sum(map(lambda a, b: (float(a) - float(b)) ** 2,
6                  x[1], y)) ** (1 / 2)
7

```

```

8
9 def comptar(l):
10    p = {}
11    for x in l:
12        p.setdefault(x, 0)
13        p[x] += 1
14    return p
15
16 def classify(t):
17     ds = list(map(deeuclidiana, centroides,
18                   [t for x in range(len(centroides))]))
19     return min([(ds[i], centroides[i][0]),
20                for i in range(len(centroides))],
21                key=lambda x: x[0])[1]
22
23 def calcularCentroides(classe):
24     filt = list(filter(lambda x: x[0] == x[1],
25                       [(clasesTrain[i], classe, train[i])
26                        for i in range(len(train))]))
27     transp = list(zip(*[filt[i][2] for i in range(len(filt))]))
28     return (classe,
29             list(map(lambda l: redueix(lambda a, b:
30                         float(a) + float(b),
31                         l) / classes[classe], transp)))
32
33 # càrrega de l'arxiu
34 l = list(map(lambda l: (l.strip()).split(','),
35              open('iris.data.txt', 'r').readlines()))
36
37 # construcció del training: 2 de cada 3
38 train = list(map(lambda x: x[1],
39                  filter(lambda v: v[0] % 3 != 0, enumerate(l)))
40 nc = len(train[0])
41 clasesTrain = list(map(lambda x: x.pop(nc - 1), train))
42
43 # construcció del test: 1 de cada 3
44 test = list(map(lambda x: x[1],
45                  filter(lambda v: v[0] % 3 == 0, enumerate(l)))
46 clasesTest = list(map(lambda x: x.pop(nc - 1), test))
47
48 # Entrenament
49 clases = comptar(clasesTrain)
50 centroides = [calcularCentroides(c) for c in clases.keys()]
51
52 # Classificació
53 prediccions = list(map(classify, test))
54
55 # Nombre de correctes
56 print('Prec.: ', len(list(filter(lambda x: x[0] == x[1],
57                               zip(*[prediccions, clasesTest]))))
58             / len(test) * 100, '%')

```

Si apliquem el programa a l'arxiu de dades del problema “iris” de la UCI (Frank i Asunción, 2010), obtindrem una precisió del 96%. Aquest programa selecciona com a conjunt de test un de cada tres exemples de l'arxiu. La resta d'exemples pertanyerà al conjunt d'entrenament.

Repositori

Els arxius de dades i el programa estan disponibles en el repositori de l'assignatura.

4.3.3. Clustering dins de classes

Aquest algorisme està a mig camí entre els dos anteriors. En aquest cas, abordarem el problema de la classificació de les flors tenint els avantatges o inconvenients dels dos.

Consisteix a aplicar un algorisme de categorització (qualsevol serveix) per a calcular un cert nombre de centroides per a cadascuna de les classes que apareix en el conjunt d'entrenament. Una vegada fet això, utilitza el kNN seleccionant tots els centroides com a conjunt d'entrenament i aplica l'1NN per a la fase de classificació.

Implementació en Python

El programa 4.4 correspon a la implementació en Python de l'algorisme que usa *clustering* dins de classes. Com a mètode de *clustering* s'ha fet servir el *k-means*^{*}, però funcionaria amb qualsevol altre mètode. Aquest programa funcionarà per a tots els problemes amb dades numèriques sense valors absents que tinguin la classe com a última columna.

* S'ha utilitzat l'algorisme de *clustering* explícit en el subapartat 2.3.4. d'aquest mòdul.

Codi 4.4: *k-means* supervisat en Python

```

1  from functools import reduce
2  from random import randrange
3  from itertools import repeat
4
5  # paràmetres
6
7  k = 3
8  maxit = 10
9
10 # declaració de funcions
11
12 def deuclidea(x, y):
13     return sum(map(lambda a, b: (float(a) - float(b)) ** 2,
14                   x, y)) ** (1 / 2)
15
16 def comptar(l):
17     p = {}
18     for x in l:
19         p.setdefault(x, 0)
20         p[x] += 1
21     return p
22
23 def classify(t):
24     ds = [deuclidea(centroides[i][1], t)
25           for i in range(len(centroides))]
26     return min([(ds[i], centroides[i][0])
27                 for i in range(len(centroides))],
28                 key=lambda x: x[0])[1]
29
30 def nextCentr(l):
31     tot2 = list(zip(*[x[2] for x in l]))
32     num = len(tot2[0])
33
34     return list(map(lambda l: reduce(lambda a, b:
35                                         float(a) + float(b), 1)
36                  / num, list(tot2)))
37
38 def kmeans(classe, k, maxit):
39     filt = list(filter(lambda x: x[0] == x[1],
40                        [(clasesTrain[i], classe, train[i])
41                         for i in range(len(train))]))
42     conj = [filt[i][2] for i in range(len(filt))]
43     centr = [conj[randrange(len(conj))] for i in range(k)]
44     anteriors = None
45
46     for it in range(maxit):
47         propers = [min(zip(*[list(map(deuclidea,
48                           repeat(ej, k),
49                           centr)), range(k))])[1]
```

```

49     for ex in conj]
50     centr2 = [nextCentr(list(filter(
51         lambda x: x[0] == x[1],
52         zip(*[list(repeat(c, len(conj))),
53             proper, conj]))))
54         for c in range(k)]
55     if proper == anteriors: break
56     anteriors = proper
57
58 return [(classe, centr2[i]) for i in range(k)]
59
60 # càrrega de l'arxiu
61 l = list(map(lambda l: (l.strip()).split(','),
62             open('iris.data.txt', 'r').readlines()))
63
64 # construcció del training: 2 de cada 3
65 train = list(map(lambda x: x[1],
66                 filter(lambda v: v[0] % 3 != 0, enumerate(l)))
67 nc = len(train[0])
68 classesTrain = list(map(lambda x: x.pop(nc - 1), train))
69
70 # construcció del test: 1 de cada 3
71 test = list(map(lambda x: x[1],
72                 filter(lambda v: v[0] % 3 == 0, enumerate(l)))
73 classesTest = list(map(lambda x: x.pop(nc - 1), test))
74
75 # Entrenament
76 classes = comptar(classesTrain)
77 centroides = reduce(lambda x, y: x + y,
78                     [kmeans(c, k, maxit) for c in classes.keys()])
79
80 # Classificació
81 prediccions = list(map(classify, test))
82
83 # Nombre de correctes
84 print('Prec.:',
85       len(list(filter(lambda x: x[0] == x[1],
86                   zip(*[prediccions, classesTest]))))
87       / len(test) * 100, '%')

```

Si apliquem el programa a l'arxiu de dades del problema “iris” de la UCI (Frank i Asunción, 2010), amb una k de 3 i un màxim d'iteracions de 10, obtindrem una precisió del 98%. Aquest programa selecciona com a conjunt de test un de cada tres exemples de l'arxiu. La resta d'exemples pertanyerà al conjunt d'entrenament.

Repositori

Els arxius de dades i el programa estan disponibles en el repositori de l'assignatura.

4.4. Mètodes basats en regles

Aquests mètodes adquireixen regles de selecció associades a cadascuna de les classes. Donat un exemple de test, el sistema selecciona la classe que verifica algunes de les regles que determinen una de les classes.

4.4.1. Arbres de decisió

Suposem que volem fer una aplicació per a telèfons mòbils que ajudi els afecionats a la recol·lecció de bolets a destriar els bolets verinosos dels comestibles a partir de les seves propietats: forma i color del barret i color del tronc.

Un arbre de decisió és una manera de representar regles de classificació inherents a les dades, amb una estructura en arbre n -ari que particiona 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 $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$, en què cada component x correspon a $x_i = (x_{y_1}, \dots, x_{i_m})$ en què m correspon al nombre d'atributs dels exemples d'entrenament; i el conjunt d'atributs per $A = \{a_1, \dots, a_m\}$ en què $\text{dom}(a_j)$ correspon al conjunt de tots els possibles valors de l'atribut a_j , i per a qualsevol valor d'un exemple d'entrenament $x_{i_j} \in \text{dom}(a_j)$.

El procés de construcció de l'arbre és un procés iteratiu, en el qual, en cada iteració, se selecciona l'atribut que particiona millor el 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 $|\text{dom}(a_j)|$ 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 a cada conjunt de la partició la classe majoritària, 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 del tipus *atribut = valor*. Si tots els exemples del conjunt han quedat ben classificats, el convertim en node terminal amb la classe dels exemples. En cas contrari, el convertim en node intern i apliquem una nova iteració al conjunt (“reduit”) eliminant l'atribut que ha generat la partició. En cas de no quedar atributs, el convertírem en node terminal assignant 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 al qual arribem.

Exemple d'aplicació

La taula 18 és una simplificació de la taula 8. 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* i la de *bell* és *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 $bondat(gill-color) = (1 + 3 + 1)/7 = 0,71$.

Taula 18. 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

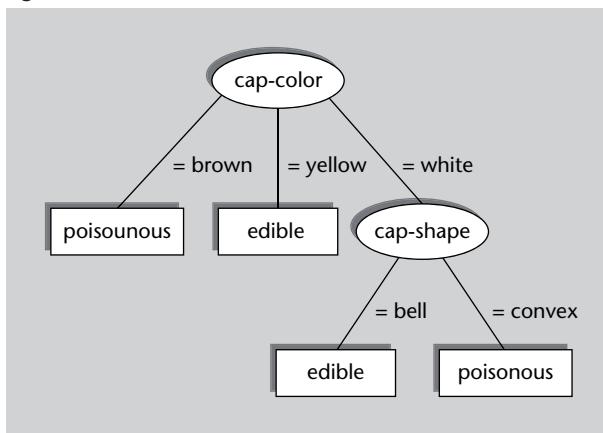
Font: problema "mushroom" del repositori UCI (Frank i Asunción, 2010).

El pas següent consisteix a seleccionar el millor atribut. Hem obtingut un empata entre els tres atributs, 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 de *white* ens queda amb les dades que mostra la taula 19. 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 28 mostra l'arbre construït en aquest procés.

Taula 19. Conjunt de la segona iteració

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

Figura 28. Arbre de decisió



Per a etiquetar un exemple de test com el que mostra la taula 20 hem de recórrer l'arbre, partint de l'arrel, 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 20. Exemple de test

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

Font: problema “mushroom” del repositori UCI (Frank i Asunción, 2010).

Anàlisi del mètode

Aquest algorisme té el gran avantatge de la facilitat d'interpretació del model d'aprenentatge. Un arbre de decisió ens dóna la 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 en el financer.

Dos dels grans inconvenients que planteja són l'elevada fragmentació de les dades en presència d'atributs amb molts valors i l'elevat cost computacional que això implica. Això fa que no sigui un mètode gaire adequat per a problemes amb grans espais d'atributs, com poden ser aquells que contenen informació lèxica, com la categorització de textos o els filtres antibrossa.

Un altre inconvenient que cal tenir en compte és que els nodes terminals corresponents a regles que donen cobertura a pocs exemples d'entrenament no produeixen estimacions fiables de les classes. Tendeixen a sobreentrenar el conjunt d'entrenament*. Per a suavitzar aquest efecte es pot utilitzar alguna tècnica de poda**. Més endavant se n'explica una.

*Aquest efecte es coneix en anglès com a *overfitting*.

** En anglès, *prunning*.

Tractament d'atributs numèrics

Suposem que ara volem fer una aplicació per a un magatzem de flors, on arriben diàriament milers de productes. Disposem d'un sistema làser que ens subministra la longitud del sèpal 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.

L'algorisme dels arbres de decisió va ser dissenyat per a tractar atributs nominals, però hi ha alternatives per al tractament d'atributs numèrics. El més comú es basa en la utilització dels punts de tall. Aquests són punts que divideixen el conjunt de valors d'un atribut en dos (els més petits i els més grans que el punt de tall).

Per a calcular el millor punt de tall d'un atribut numèric, s'ordenen els valors i s'eliminen els elements repetits. Es calculen els possibles punts de tall com la mitjana de cada dos valors consecutius. Com a últim pas es calcula el millor com aquell amb més bondat.

Hem de tenir en compte que el tractament d'atributs numèrics genera arbres (i decisions) binaris. Això implica que en els nivells següents de l'arbre haurem de tornar a processar l'atribut numèric que acabem de seleccionar, a diferència dels atributs nominals.

Exemple del càlcul del millor punt de tall d'un atribut numèric

A continuació s'exposa el càlcul del millor punt de tall per a l'atribut *sepal-length* del conjunt de dades que mostra la taula 13. La taula 21 mostra aquests càlculs: les columnes 1 i 2 mostren la classe i l'atribut ordenats per l'atribut, la 3 mostra els valors sense els repetits, la 4 els punts de tall com la mitjana de cada dos valors consecutius i l'última columna mostra les bondats dels punts de tall. El millor punt de tall és 5,35, amb una bondat del 66,7%.

Taula 21. Càlculs per al millor punt de tall

classe	sepal-length	sense els repetits	punts de tall	bondat
setosa	4,9			
virgíñica	4,9	4,9	5	(1 + 2)/6 = 0,5
setosa	5,1	5,1	5,35	(2 + 2)/6 = 0,67
versicolor	5,6	5,6	5,85	(2 + 1)/6 = 0,5
versicolor	6,1	6,1	6,85	(2 + 1)/6 = 0,5
virgíñica	7,6	7,6		

Algorisme ID3

L'algorisme ID3 és una modificació dels arbres de decisió que utilitza el guany d'informació* com a mesura de bondat per a analitzar els atributs. El concepte de guany d'informació és un concepte que està basat en l'entropia. L'ús de l'entropia tendeix a penalitzar més els conjunts barrejats.

* En anglès, *information gain*.

L'entropia és una mesura de la teoria de la informació que quantifica el desordre. Així, donat un conjunt S , la seva fórmula està determinada per:

$$H(S) = - \sum_{y \in I} p(y) \log_2(p(y))$$

en què $p(y)$ és la proporció d'exemples de S que pertany a la classe y . L'entropia serà mínima (0) quan en S hi ha una sola classe i màxima (1) quan el conjunt és totalment aleatori.

A tall d'exemple, l'entropia del conjunt que mostra la taula 18 està determinada per:

$$H(S) = -\frac{3}{7} \log_2 \frac{3}{7} - \frac{4}{7} \log_2 \frac{4}{7} = 0,985$$

La fórmula del guany d'informació per al conjunt S i l'atribut a_j queda com:

$$G(S, a_j) = H(S) - \sum_{v \in a_j} p(v)H(S_v)$$

en què $p(v)$ és la proporció d'exemples de S que tenen el valor v per a l'atribut a_j ; i S_v és el subconjunt de S dels exemples que prenen el valor v per a l'atribut a_j .

A tall d'exemple, el guany d'informació del conjunt que mostra la taula 18 i l'atribut *cap_shape* està determinat per:

$$G(S, cs) = H(S) - \frac{5}{7}H(S_{cs=convex}) - \frac{2}{7}H(S_{cs=bell}) = 0,292$$

en què cs és *cap_shape*,

$$H(S_{cs=convex}) = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} = 0,971$$

i

$$H(S_{cs=bell}) = -\frac{0}{2} \log_2 \frac{0}{2} - \frac{2}{2} \log_2 \frac{2}{2} = 0$$

Hi ha altres variants d'arbres de decisió que utilitzen altres mesures per a computar la bondat dels atributs. Algunes són l'ús de la variància quan ens trobem atributs numèrics. Una altra mesura utilitzada és la informació mútua*, que està relacionada amb l'entropia creuada.

* En anglès, *mutual information*.

Podar dels arbres de decisió

Vegeu també

Sobre la informació mútua podeu veure el subapartat 3.1.3.

L'objectiu de la poda dels arbres de decisió és obtenir arbres que no tinguin en les fulles regles que afectin pocs exemples del conjunt d'entrenament. És desitjable no forçar la construcció de l'arbre per a evitar el sobreentrenament*.

* En anglès, *overfitting*.

La primera aproximació per a abordar la solució d'aquest problema consisteix a establir un llindar de reducció de l'entropia. És a dir, pararem una branca quan la disminució d'entropia no superi aquest llindar. Aquesta aproximació planteja el problema que pot ser que no es redueixi en un cert pas però sí en el següent.

Una altra aproximació, que soluciona el problema de l'anterior, i que és la que se sol utilitzar, consisteix a construir tot l'arbre i, en una segona fase, eliminar els nodes superflus (poda de l'arbre*). Per a fer aquest procés de poda recorrem l'arbre de manera ascendent (de les fulles a l'arrel) i anem mirant si cada parell de nodes incrementa l'entropia per sobre d'un cert llindar si els ajuntéssim. Si això passa, desfem la partió.

* En anglès, *prunning*.

Implementació en Python

El programa 4.5 correspon a la implementació en Python dels arbres de decisió bàsics amb atributs nominals. Aquest programa funcionarà per a tots els problemes amb dades nominals sense valors absents que tinguin la classe com a primera columna.

Codi 4.5: Arbres de decisió en Python

```

1 # declaració de funcions
2
3 def bondad(classes , conjunt):
4     p = {}
5     for i in range(len(classes)):
6         p.setdefault(conjunt[i] , {})
7         p[conjunt[i]].setdefault(classes[i] , 0)
8         p[conjunt[i]][classes[i]] += 1
9
10    return sum([max([(val , p[atr][val])
11                    for val in p[atr].keys()])[1]
12                  for atr in p.keys()])
13
14 def classeMF(classes):
15     p = {}
16     for x in classes:
17         p.setdefault(x , 0)
18         p[x] += 1
19     return max([(p[c1] , c1) for c1 in p.keys()])[1]
20
21 def iteracio2(c):
22     c.pop(1)
23     return (c[0][0] , iteracio(list(c[2]) , list(zip(*c[1]))))
24
25 def iteracio(cl , cj):
26     l = sorted(set(cl))
27     if len(l)==1:
28         return ("classe" , l[0])
29     else:
30         (b , col) = max([(bondat(cl , cj[i]) , i)
31                           for i in range(len(cj))])
32         l = cj.pop(col)
33         lu = sorted(set(l))
34         cj = list(zip(*cj))
35         if len(cj) == 0:
36             l = [list(filter(lambda x: x[0] == x[1] , y))
37                   for y in [[(val , l[i] , cl[i])
38                               for i in range(len(l))]]
39                               for val in lu]]
40
        return ("atr" , col ,
```

```

41          [(lp[0][0],
42             ("classe", classeMF(list(list(zip(*lp))[2]))))
43             for lp in l])
44     elif b == len(cl):
45         l = [list(filter(lambda x: x[0] == x[1], y))
46             for y in [[(val, l[i], cl[i])
47                 for i in range(len(l))]
48                 for val in lu]]
49     return ("atr", col,
50             [(lp[0][0],
51               ("classe", classeMF(list(list(zip(*lp))[2]))))
52               for lp in l])
53   else:
54     l = [list(filter(lambda x: x[0] == x[1], y))
55             for y in [[(val, l[i], list(cj[i]), cl[i])
56                 for i in range(len(l))]
57                 for val in lu]]
58   return ('atr', col, [iteracion2(list(zip(*x)))
59                         for x in l])
60
61 def testEx(ex, arbre):
62     if isinstance(arbre, tuple):
63         if arbre[0] == "classe":
64             return arbre[1]
65         elif arbre[0] == "atr":
66             valor = ex.pop(arbre[1])
67             arbol = list(filter(lambda x: x[0][0] == x[1],
68                             [(1, valor) for l in arbre[2]]))
69             if len(arbre)==0:
70                 return None
71             else:
72                 return testEx(ex, arbre[0][0][1])
73         else:
74             return None
75     else:
76         return None
77
78 # càrrega de l'arxiu
79 l = list(map(lambda l: (l.strip()).split(','), 
80             open('mushroom.data.txt', 'r').readlines()))
81
82 # construcció del training: 2 de cada 3
83 train = list(map(lambda x: x[1],
84                 filter(lambda v: v[0] % 3 != 0, enumerate(l))))
85 classesTrain = list(map(lambda x: x.pop(0), train))
86
87 # construcció del test: 1 de cada 3
88 test = list(map(lambda x: x[1],
89                 filter(lambda v: v[0] % 3 == 0, enumerate(l))))
90 classesTest = list(map(lambda x: x.pop(0), test))
91
92 # Construcció de l'arbre
93 arbre = iteracio(classesTrain, list(zip(*train)))
94 print(arbre)
95
96 # Classificació
97 prediccions = [testEx(ex, arbre) for ex in test]
98
99 # Nombre de correctes
100 print('Prec.: ', len(list(filter(lambda x: x[0] == x[1],
101                               zip(*[prediccions, classesTest]))))/
102                               len(test) * 100, '%')

```

Si apliquem el programa a l'arxiu de dades del problema “mushroom” de la UCI (Frank i Asunción, 2010) obtindrem una precisió del 100%. El programa mostra l'arbre generat com a sortida amb format parentitzat. Aquest programa selecciona com a conjunt de test un de cada tres exemples de l'arxiu. La resta d'exemples pertanyerà al conjunt d'entrenament.

Repositori

Els arxius de dades i el programa estan disponibles en el repositori de l'assignatura.

4.4.2. AdaBoost

Suposem que volem fer una aplicació per a telèfons mòbils que ajudi els afecionats a la recol·lecció de bolets a destriar els bolets verinosos dels comestibles a partir de les seves propietats: forma i color del barret.

Aquest mètode està específicament dissenyat per a combinar regles. Es basa en la combinació lineal de moltes regles molt senzilles però molt precises (anomenades *regles febles**), per a crear un classificador molt robust amb un error arbitràriament baix en el conjunt d'entrenament. Aquestes regles febles s'aprenen seqüencialment mantenint una distribució de pesos sobre els exemples d'entrenament. Aquests pesos es van actualitzant a mesura que anem adquirint noves regles. Aquesta actualització depèn de la dificultat d'aprenentatge dels exemples d'entrenament. És a dir, els pesos dels exemples seran directament proporcionals a la seva dificultat d'aprenentatge, i per tant, l'adquisició de noves regles s'anirà dirigint als exemples amb més pes (i dificultat).

* En anglès, *weak rules* o *weak hypotheses*.

Lectura complementària

L'algorisme AdaBoost va ser proposat a: Y. Freund; R. E. Schapire (1997). “A Decision-theoretic Generalization of On-line Learning and an Application to Boosting”. *Journal of Computer and System Sciences* (vol. 1, núm. 55).

AdaBoost bàsic

L'AdaBoost és un algorisme que pretén obtenir una regla de classificació molt precisa combinant molts classificadors febles, cadascun dels quals obté una precisió moderada. Aquest algorisme treballa eficientment amb espais d'atributs molt grans i ha estat aplicat amb èxit a molts problemes pràctics. A continuació es mostra el pseudocodi de l'algorisme:

Referència bibliogràfica

Y. Freund; R. E. Schapire (1999). “A Short Introduction to Boosting”. *Journal of Japanese Society for Artificial Intelligence* (vol. 5, núm. 14, pàg. 771-780).

algorisme AdaBoost

entrada: $S = \{(x_i, y_i), \forall i = 1..m\}$

S és el conjunt d'exemples d'entrenament
D_1 és la distribució inicial de pesos
m és el nombre d'exemples d'entrenament
$y_i \in \{-1, +1\}$

$D_1(i) = \frac{1}{m}, \forall i = 1..m$

Es fan T iteracions:

per a $t := 1$ fins a T fer

S'obté la hipòtesi feble $h_t : X \rightarrow \{-1, +1\}$

$h_t = \text{ObtenirHipòtesiFeble}(X, D_t)$

$\epsilon_t = \sum_{i: h_t(x_i) \neq y_i} D_t(i)$

$\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$

S'actualitza la distribució D_t

$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}, \forall i = 1..m$

Z_t és un factor de normalització tal que D_{t+1} sigui una

distribució, per exemple: $Z_t = \sum_{i=1}^m D_t(i) \exp(-\alpha_t y_i h_t(x_i))$

fi per a

retorna la combinació d'hipòtesi: $f(x) = \sum_{t=1}^T \alpha_t h_t(x)$

fi AdaBoost

Les hipòtesis febles s'aprenen seqüencialment. En cada iteració s'aprèn una regla que s'esbiaixa per a classificar els exemples amb més dificultats per part del conjunt de regles precedents. AdaBoost manté un vector de pesos com una distribució D_t sobre els exemples. En la iteració t , l'objectiu de l'algorisme és trobar una hipòtesi feble, $h_t : X \rightarrow \{-1, +1\}$, amb un error moderadament baix pel que fa a la distribució de pesos D_t . En aquest marc, les hipòtesis febles $h_t(x)$ donen com a prediccions valors reals que corresponen a valors de confiança. Inicialment, la distribució de pesos D_1 és uniforme, i en cada iteració, l'algorisme de *boosting* incrementa (o disminueix) exponencialment els pesos $D_t(i)$ en funció de si $h_t(x_i)$ fa una predicció bona (o dolenta). Aquest canvi exponencial depèn de la confiança $|h_t(x_i)|$. La combinació final d'hipòtesis, $h_t : X \rightarrow \{-1, +1\}$, calcula les seves prediccions ponderant amb pesos els vots de les diferents hipòtesis febles,

$$f(x) = \sum_{t=1}^T \alpha_t \cdot h_t(x)$$

Per a cada exemple x , el signe de $f(x)$ s'interpreta com la classe predicta (l'AdaBoost bàsic treballa només amb dues classes de sortida, -1 o $+1$), i la magnitud $|f(x)|$ com una mesura de la confiança de la predicció. Aquesta funció es pot usar per a classificar nous exemples o per a fer-ne un rànquing en funció d'un grau de confiança.

Exemple d'aplicació

L'algorisme AdaBoost té dos paràmetres per determinar: el nombre màxim d'iteracions i la manera de construcció de les regles febles. En aquest subapartat veurem l'aplicació de l'algorisme escollint com a conjunt d'entrenament les dades de la taula 22 i com a test el de la taula 23. Fixem el nombre màxim d'iteracions a 5 i com a regles febles, regles de l'estil: *atribut = valor* i *atribut ≠ valor*.

Taula 22. Conjunt d'entrenament

class	cap-shape	cap-color
+1	convex	brown
-1	convex	yellow
-1	bell	white
+1	convex	white

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

Taula 23. Exemple de test

class	cap-shape	cap-color
-1	bell	yellow

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

Com a primer pas de l'algorisme inicialitzem els 4 elements del vector de pesos D_1 a 0,25. A continuació, comencem el procés iteratiu. El primer és calcular la regla feble. La taula 24 mostra les diferents regles aplicables* i el seu error. La regla $cs = bell$ assignarà les prediccions: -1, -1, +1 i -1, respectivament. Per a calcular l'error sumem el pes $D_t(i)$ dels exemples les prediccions dels quals siguin diferents a les seves classes. En aquest cas sumaríem el pes del primer, segon i quart exemples (en total, 0,75). La resta de casos els calcularíem de manera similar.

* En què cs correspon a $cap-shape$ i cc a $cap-color$.

Taula 24. Regles febles

Regla	Error
$cs = bell$	0,75
$cs \neq bell$	0,25
$cs = convex$	0,25
$cs \neq convex$	0,75
$cc = brown$	0,25
$cc \neq brown$	0,75
$cc = white$	0,5
$cc \neq white$	0,5
$cc = yellow$	0,75
$cc \neq yellow$	0,25

Escollim com a regla feble una de les que tinguin error mínim (en l'exemple hem escollit $cs \neq bell$). Així, l'error de la primera iteració és $\epsilon_1 = 0,25$ i per tant $\alpha_1 = \frac{1}{2} \ln(\frac{1-\epsilon_1}{\epsilon_1}) = 0,5493$.

El pas següent consisteix a actualitzar el vector de pesos. Comencem per calcular $D_1(i) \exp(-\alpha_1 y_i h_1(x_i))$. La taula 25 mostra el valor d'aquesta expressió per als 4 exemples i el valor de tots els components. L'última columna mostra els valors finals del vector de pesos per a la segona iteració.

Taula 25. Càlcul de D_2

i	$D_1(i)$	α_1	y_i	$h_1(x_i)$	numerador	$D_2(i)$
1	0,25	0,5493	+1	+1	0,1443376	0,1667
2	0,25	0,5493	-1	+1	0,4330127	0,5
3	0,25	0,5493	-1	-1	0,1443376	0,1667
4	0,25	0,5493	+1	+1	0,1443376	0,1667

La resta d'iteracions es calcularien de manera similar. La taula 26 mostra els resultats de les 5 iteracions de l'entrenament.

Taula 26. Evolució dels pesos

t	α_t	reglat	D_t
1	0,549	$cs \neq bell$	(0,25,0,25,0,25,0,25)
2	0,805	$cc = brown$	(0,167,0,5,0,167,0,167)
3	1,099	$cc \neq yellow$	(0,999,0,3,0,099,0,499)
4	0,805	$cs \neq bell$	(0,056,0,167,0,5,0,278)
5	0,805	$cc = brown$	(0,033,0,5,0,3,0,167)
6			(0,019,0,3,0,18,0,5)

El classificador dóna una predicció correcta sobre l'exemple que mostra la taula 23. Per a obtenir la predicció hem de calcular:

$$\begin{aligned} \text{signe}(\sum_{t=1}^5 \alpha_t h_t(x)) &= \\ &= \text{signe}(0,549 \times (-1) + 0,805 \times (-1) + \\ &\quad + 1,099 \times (-1) + 0,805 \times (-1) + 0,805 \times (-1)) = \\ &= \text{signe}(-4,063) = -1 \end{aligned}$$

Anàlisi del mètode

Per a garantir l'estabilitat de l'algorisme ens hem d'assegurar que l'error es mantingui entre 0 i 0,5. Si l'error puja d'aquesta mesura, no en podem assegurar la convergència.

Aquest algorisme s'ha aplicat amb èxit a molts problemes pràctics, incloent-hi la categorització de textos o el filtre antibrossa.

Els avantatges que ens aporta aquest mètode són diversos. En primer lloc, té pocs paràmetres: el nombre màxim d'iteracions i la forma de les hipòtesis febles. La implementació d'aquest algorisme és relativament senzilla.

D'altra banda, en finalitzar el procés, ens deixa informació molt útil. De les regles generades es pot extreure coneixement que pot ser interpretat per humans. Els pesos finals indiquen quins exemples no s'han pogut modelitzar. Això ens dóna informació de possibles *outliers* o possibles errors d'etiquetatge.

El principal inconvenient que té és el seu cost computacional. S'han dedicat esforços a tractar aquest tema, com la versió LazyBoosting,* en què en cada iteració només s'explora un subconjunt aleatori de les possibles regles febles. En aquest treball es reporta una reducció substancial del cost computacional amb una reducció mínima dels resultats.

* G. Escudero; L. Márquez; G. Rigau (2000). "Boosting Applied to Word Sense Disambiguation". A: *Proceedings of the 12th European Conference on Machine Learning*.

L'AdaBoost és un algorisme teòricament ben fonamentat. L'error de generalització de la hipòtesi final pot ser fitat en termes de l'error d'entrenament. Aquest algorisme està basat en la maximització del marge, la qual cosa fa que tingui moltes similituds amb les màquines de vectors de suport. Aquests dos conceptes estan tractats amb més profunditat en el subapartat 4.5.

L'algorisme bàsic només permet classificar entre dues classes. Per a tractar problemes multiclass, en lloc de binaris hem d'utilitzar la variant AdaBoost.MH.

Lectura complementària

El AdaBoost.MH va ser proposat en: R. E. Schapire; Y. Singer (2000). "Boostexter: A Boosting-based System for Text Categorization". *Machine Learning* (vol. 39, núm. 2-3)

Implementació en Python

El programa 4.6 correspon a la implementació en Python de l'AdaBoost binari bàsic amb atributs nominals. Aquest programa funcionarà per a tots els problemes amb dades nominals sense valors absents que tinguin la classe com a primera columna i codificada com -1 i +1. Té definit el màxim d'iteracions en 50.

Codi 4.6: AdaBoost en Python

```

1  from itertools import repeat
2  from math import log
3  from math import exp
4
5  # paràmetres
6  T = 50
7
8  # declaració de funcions
9
10 def hti(atvl, val, igual):
11     if (igual and (val == atvl)) or (not igual and (val != atvl)):
12         pred = +1
13     else:
14         pred = -1
15     return pred
16
17 def error(Dt, cl, atvl, val, igual):
18     s = 0
19     for i in range(len(cl)):
20         if hti(atvl[i], val, igual) != int(cl[i]):
21             s += Dt[i]
22     return s
23
24 def WeakLearner(Dt, tr):
25     ll = []
26     for atr in range(len(tr)-1):
27         for val in sorted(set(tr[atr+1])):
28             et = error(Dt, tr[0], tr[atr+1], val, True)
29             ll.append([et, atr+1, val, True])
30             et = error(Dt, tr[0], tr[atr+1], val, False)
31             ll.append([et, atr+1, val, False])
32     return min(ll)
33
34 def testEx(ex, model):
35     return sum([regla[0] * hti(ej[regla[2]], regla[3], regla[4])
36                 for regla in model])
37
38 # càrrega de l'arxiu
39 l = list(map(lambda l: (l.strip()).split(','), 
40             open('mushroom.adaboost.txt', 'r').readlines()))
41
42 # construcció del training: 2 de cada 3
43 train = list(map(lambda x: x[1],
44                  filter(lambda v: v[0] % 3 != 0, enumerate(l))))
45
46 # construcció del test: 1 de cada 3
47 test = list(map(lambda x: x[1],
48                  filter(lambda v: v[0] % 3 == 0, enumerate(l))))
49
50 # Entrenament
51 tt = list(zip(*train))
52 m = len(train)

```

```

53 Dt = list(repeat(1/m, m))
54 modelo = []
55 for i in range(T):
56     [et, atr, val, igual] = WeakLearner(Dt, tt)
57     if et > 0.5:
58         print("Error: et_out_of_range!")
59         break
60     if et == 0:
61         print("Està tot ben classificat!")
62         break
63     alfat = 1 / 2 * log((1 - et) / et)
64     Dt2 = [Dt[j] * exp(-alfat * hti(tt[atr][j], val, igual) *
65                         float(train[j][0])) for j in range(m)]
66     s = sum(Dt2)
67     Dt = [Dt2[j]/s for j in range(m)]
68     model.append((alfat, et, atr, val, igual))
69 print(len(model), "regles_apreses!")
70
71 # Classificació
72 prediccions = [testEx(ex, model) for ex in test]
73
74 # Nombre de correctes
75 print('Prec.:', len(list(filter(
76     lambda x: x[0] * float(x[1][0]) > 0,
77     zip(*[prediccions, test])))) /
78     len(test) * 100, '%')

```

Si apliquem el programa a l'arxiu de dades del problema “mushroom” de la UCI (Frank i Asunción, 2010) obtindrem una precisió del 99,889%. Aquest programa selecciona com a conjunt de test un de cada tres exemples de l'arxiu. La resta d'exemples pertanyerà al conjunt d'entrenament.

Repositori

Els arxius de dades i el programa estan disponibles en el repositori de l'assignatura.

4.5. Classificadors lineals i mètodes basats en *kernels*

En aquest subapartat començarem per veure un algorisme equivalent al classificador lineal que es descriu en el subapartat 4.3.2. En la versió d'aquest subapartat substituirem la distància euclidiana pel producte escalar. Aquesta modificació ens permetrà introduir l'ús dels *kernels*: una tècnica molt utilitzada en l'última dècada per al tractament de conjunts no lineals amb algoritmes lineals. Per a finalitzar el subapartat veurem l'algorisme de les màquines de vectors de suport; possiblement l'algorisme d'aprenentatge supervisat més important del moment.

4.5.1. Classificador lineal basat en producte escalar

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 sobre 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.

Un classificador lineal (binari) és un hiperplà en un espai de dimensió n d'atributs que pot ser representat per un vector de pesos w i un llindar* b que està relacionat amb la distància de l'hiperplà a l'origen: $h(x) = \text{signe}(\langle w, x \rangle + b)$ (en què $\langle \cdot, \cdot \rangle$ representa el producte escalar). Cada component del vector de pesos es correspon amb un dels atributs.

* En anglès, *bias*.

L'exemple més simple de classificador lineal és el mètode basat en els centroides descrit en el subapartat 4.3.2. Una vegada calculats els centroides p i n com el dels exemples amb classes +1 i -1, respectivament, podem expressar la predicció d'un nou exemple com:

$$h(x) = \begin{cases} +1, & \text{si } de(x,p) < de(x,n) \\ -1, & \text{en cas contrari} \end{cases}$$

en què $de(x,p)$ correspon a la distància euclidiana entre x i p .

Sabent que la distància euclidiana es pot expressar com $\|x - p\|^2$, l'expressió anterior es pot reformular com:

$$h(x) = \text{signe}(\|x - n\|^2 - \|x - p\|^2)$$

Tenint en compte que $\|a - b\|^2 = \langle a, a \rangle + \langle b, b \rangle - 2\langle a, b \rangle$, desenvolupant l'argument de la funció signe obtenim:

$$\begin{aligned} \|x - n\|^2 - \|x - p\|^2 &= \\ &= \langle x, x \rangle + \langle n, n \rangle - 2\langle x, n \rangle - \langle x, x \rangle - \langle p, p \rangle + 2\langle x, p \rangle = \\ &= 2\langle x, p \rangle - 2\langle x, n \rangle - \langle p, p \rangle + \langle n, n \rangle = \\ &= 2\langle x, p - n \rangle - \langle p, p \rangle + \langle n, n \rangle \end{aligned}$$

per finalitzar aquest desenvolupament, obtindrem la relació amb w i b :

$$\begin{aligned} h(x) &= \text{signe}(2\langle x, p - n \rangle - \langle p, p \rangle + \langle n, n \rangle) = \\ &= \text{signe}(\langle x, p - n \rangle - \frac{1}{2}(\langle p, p \rangle - \langle n, n \rangle)) = \text{signe}(\langle w, x \rangle - b) \end{aligned}$$

I per tant:

$$w = p - n$$

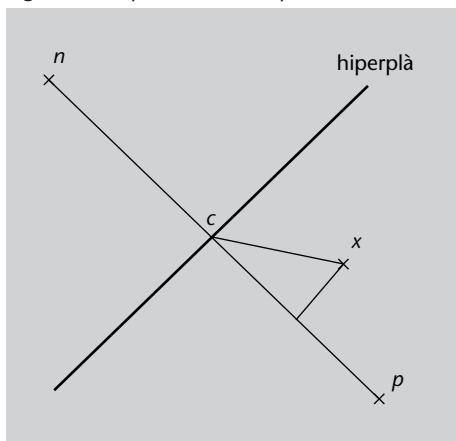
$$b = \frac{1}{2}(\langle p, p \rangle - \langle n, n \rangle)$$

Producte escalar

El producte escalar és la projecció d'un vector sobre l'altre. Si tenim un punt x com el de la figura 29 i els dos centroides p per als exemples de la classe $+1$, n per als -1 i c com el punt mitjà entre tots dos, podem aprofitar el producte escalar per a obtenir la predicció. La predicció per a aquest punt estarà determinada pel signe del producte escalar dels vectors $c \rightarrow p$ i $c \rightarrow x$:

$$h(x) = \text{signe}(\langle p - c, x - c \rangle)$$

Figura 29. Representació del producte escalar



Desenvolupant aquesta expressió obtenim:

$$y = \text{signe}(\langle p - c, x \rangle + \langle c, c - p \rangle) = \text{signe}(\langle w, x \rangle + b)$$

I per tant:

$$w = p - c = p - \frac{p+n}{2} = \frac{p-n}{2}$$

$$b = \langle c, c - p \rangle = \left\langle \frac{p+n}{2}, \frac{p+n}{2} - p \right\rangle = \left\langle \frac{p+n}{2}, \frac{n-p}{2} \right\rangle =$$

$$= \frac{\langle p, n \rangle - \langle p, p \rangle + \langle n, n \rangle - \langle p, n \rangle}{4} = \frac{\langle n, n \rangle - \langle p, p \rangle}{4}$$

La magnitud del producte escalar ens està donant una mesura de la distància del punt per classificar a l'hiperplà, que es pot interpretar com una mesura de la confiança de la predicció.

Fixeu-vos que amb els dos desenvolupaments hem arribat al mateix resultat*. Hi ha una relació directa entre les distàncies euclidianes i el producte escalar.

* Noteu que multiplicar per un valor positiu no canvia el resultat de la funció signe.

Exemple d'aplicació

La taula 27 mostra un conjunt d'entrenament en el qual hem de classificar flors a partir de les seves propietats. Els centroides es calculen a partir de les mitjanes dels diferents atributs i corresponen, respectivament, a: $p = (5,0, 3,25, 1,4, 0,2)$ i $n = (5,85, 2,9, 4,15, 1,35)$.

Taula 27. Conjunt d'entrenament

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

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

Una vegada obtinguts els centroides, passem a obtenir els valors de l'hiperplà w i b .

$$w = \frac{p - n}{2} = (-0,425, 0,175, -1,375, -0,575)$$

$$b = \frac{\langle n, n \rangle - \langle p, p \rangle}{4} = 6,02875$$

A partir d'aquest moment, si volem classificar l'exemple que mostra la taula 28 avaluem la fórmula:

$$\begin{aligned} y &= \text{signe}(\langle w, x \rangle + b) = \\ &= \text{signe}(\langle (-0,425, 0,175, -1,375, -0,575), (4,9, 3,1, 1,5, 0,1) \rangle + 6,02875) = \\ &= \text{signe}(2,36875) = +1 \end{aligned}$$

Taula 28. Exemple de test

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

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

Anàlisi del mètode

Aquest tipus de mètode ha estat molt utilitzat en l'àrea de la recuperació de la informació*, ja que és una manera simple, eficient i efectiva de construir models per a la categorització de textos. Hi ha molts algorismes per a entrenar aquest tipus de classificadors (perceptró, Widrow-Hoff, Winnow...). Un algoritme d'aquesta família ha guanyat molta importància en l'última dècada: les màquines de vectors de suport.

* En anglès, *information retrieval*.

Implementació en Python

El programa 4.7 correspon a la implementació en Python del classificador lineal binari basat en els centroides que utilitza el producte escalar en lloc de la distància euclidiana. Aquest programa funcionarà per a tots els problemes amb dades numèriques sense valors absents que tinguin la classe com a última columna i codificada com a -1 i +1.

Codi 4.7: Lineal amb producte escalar en Python

```

1 from functools import reduce
2
3 # declaració de funcions
4
5 def prodEscalar(a, b):
6     return sum(a[i]*b[i] for i in range(len(a)))
7
8 def h(t, w, b):
9     y = prodEscalar(w, t) - b
10    if y > 0:
11        return +1
12    else:
13        return -1
14
15 # càrrega d'arxiu
16 l = list(map(lambda l: (l.strip()).split(','), 
17             open('iris.data.txt', 'r').readlines()))
18
19 # construcció del training: 2 de cada 3
20 train = list(map(lambda x: x[1],
21                  filter(lambda v: v[0] % 3 != 0, enumerate(l))))
22
23 # construcció del test: 1 de cada 3
24 test = list(map(lambda x: x[1],
25                  filter(lambda v: v[0] % 3 == 0, enumerate(l))))
26 classesTest = list(map(lambda x: int(x.pop(len(x)-1)), test))
27
28 # Entrenament
29 tp = list(zip(*filter(
30     lambda x: x[len(train[0])-1] == '+1', train)))
31 tp.pop(len(tp)-1)
32 tn = list(zip(*filter(
33     lambda x: x[len(train[0])-1] == '-1', train)))
34 tn.pop(len(tn)-1)
35 p = list(map(lambda l: reduce(lambda x, y: float(x)+float(y), 1) /
36             len(1), tp))
37 n = list(map(lambda l: reduce(lambda x, y: float(x)+float(y), 1) /
38             len(1), tn))
39 w = list(map(lambda a, b: a - b, p, n))
40 b = (prodEcalar(p, p) - prodEcalar(n, n)) / 2
41 # Test
42 prediccions = [h(list(map(lambda x: float(x), t)), w, b)
43                 for t in test]
44
45 # Nombre de correctes
46 print('Prec.: ', len(list(filter(lambda x: x[0] == x[1],
47                                     zip(*[prediccions, classesTest])))) /
48             len(test) * 100, '%')

```

Si apliquem el programa a l'arxiu de dades del problema “iris” de la UCI (Frank i Asunción, 2010), utilitzant la classe *Iris-setosa* com a classe +1 i *Iris-versicolor* com a -1, obtindrem una precisió del 100%. Aquest programa selecciona com a conjunt de test un de cada tres exemples de l'arxiu. La resta d'exemples pertanyerà al conjunt d'entrenament.

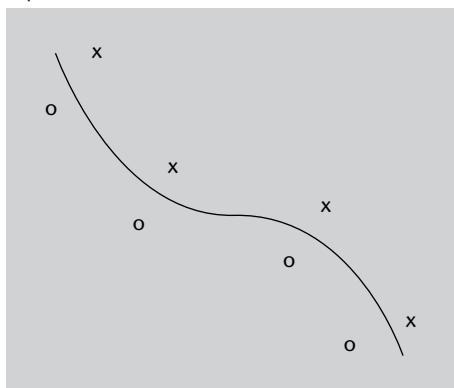
Repositori

Els arxius de dades i el programa estan disponibles en el repositori de l'assignatura.

4.5.2. Classificador lineal amb *kernel*

Els algorismes lineals com l'anterior estan pensats per a classificar conjunts linealment separables. En el cas d'un espai d'atributs que es pugui representar en el pla, l'hiperplà corresponia a una línia recta. En molts problemes reals no se sol donar aquesta circumstància; són conjunts per als quals no hi ha hiperplans lineals que els puguin separar. La figura 30 mostra un exemple d'un d'aquests casos. S'han dedicat molts esforços en la bibliografia a crear algorismes no lineals per a abordar aquest tipus de problemes. El gran inconvenient que plantegen la majoria d'aquests mètodes és l'elevat cost computacional.

Figura 30. Exemple de conjunt no linealment separable



Lectures complementàries

J. Shawe-Taylor; N. Cristianini (2004). *Kernel Methods for Pattern Analysis*. RU: Cambridge University Press.

En les dues últimes dècades s'han usat amb molta efectivitat els anomenats *kernels* (o funcions nucli). L'ús d'aquests permet atacar problemes no lineals amb algorismes lineals. L'estrategia que se segueix consisteix a projectar les dades a altres espais d'atributs en els quals sí que hi hagi un hiperplà lineal que els puguin separar*. Amb això aconseguim el cost computacional dels algorismes lineals amb només un petit increment que comporta la projecció de les dades a un altre espai d'atributs.

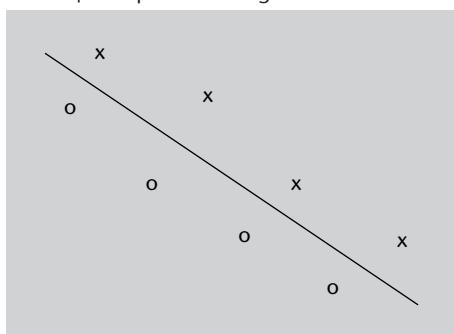
Així, si coneixem una funció ϕ que projecti els punts de la figura 30 a un espai d'atributs com el que mostra la figura 31, podrem trobar un hiperplà lineal en el nou espai que separi els dos tipus de punts.

Vegeu també

En el subapartat 4.5.4. es dóna una interpretació geomètrica dels kernels i el seu ús.

* En la bibliografia se sol citar com el truc o estratègema del *kernel*, en anglès *kernel trick*.

Figura 31. Exemple de projecció amb certa funció ϕ dels punts de la figura 30



De fet, no es treballa amb les funcions tipus ϕ , sinó amb les anomenades *funcions kernel*, $\kappa(x,z) = \langle \phi(x), \phi(z) \rangle$. Quan es treballa amb *kernels* se sol utilitzar una estructura modular en la qual primer es projecten els punts (exemples d'entrenament) de dos en dos amb la funció κ , i es crea així una matriu quadrada. Aquesta matriu es coneix com a *matriu kernel**. Els algorismes d'aprenentatge poden soler actuar sobre el *kernel* κ o sobre aquesta matriu directament. La notació estàndard per a mostrar les matrius *kernel* és:

$$\begin{pmatrix} \kappa(x_1, x_1) & \kappa(x_1, x_2) & \cdots & \kappa(x_1, x_l) \\ \kappa(x_2, x_1) & \kappa(x_2, x_2) & \cdots & \kappa(x_2, x_l) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(x_l, x_1) & \kappa(x_l, x_2) & \cdots & \kappa(x_l, x_l) \end{pmatrix}$$

en què l correspon al nombre d'exemples d'entrenament. Les matrius *kernel* han de ser simètriques i semidefinides positivament. De manera equivalent, la seva funció *kernel* associada ha de complir la mateixa propietat.

A partir d'aquest moment anem a desenvolupar la formulació tenint en compte que coneixem una suposada funció *kernel* κ , però no en coneixem la funció ϕ relativa. Com en el primer desenvolupament de l'algorisme anterior, suposem que p i n corresponen als centroides dels exemples amb classes +1 i -1 respectivament, i $|p|$ i $|n|$ al nombre d'exemples de les classes +1 i -1, respectivament. La predicció d'un nou exemple x està determinada per:

$$h(x) = \text{signe}(\|\phi(x) - \phi(n)\|^2 - \|\phi(x) - \phi(p)\|^2)$$

per a continuar desenvolupant aquesta expressió hem de tenir en compte que $\kappa(n,n) = \langle \phi(n), \phi(n) \rangle$ és equivalent a la mitjana de les projeccions de dues en dues dels elements amb classe -1. Amb p i els exemples de la classe +1 funciona de manera similar. Si desenvolupem l'argument de la funció signe tenint en compte aquestes observacions, obtenim:

$$\begin{aligned} \|\phi(x) - \phi(n)\|^2 - \|\phi(x) - \phi(p)\|^2 &= \\ \kappa(x,x) + \frac{1}{|n|^2} \sum_{\{i|y_i=-1\}} \sum_{\{j|y_j=-1\}} \kappa(x_i, x_j) - \frac{2}{|n|} \sum_{\{i|y_i=-1\}} \kappa(x, x_i) - \\ - \kappa(x,x) - \frac{1}{|p|^2} \sum_{\{i|y_i=+1\}} \sum_{\{j|y_j=+1\}} \kappa(x_i, x_j) + \frac{2}{|p|} \sum_{\{i|y_i=+1\}} \kappa(x, x_i) &= \\ = \frac{2}{|p|} \sum_{\{i|y_i=+1\}} \kappa(x, x_i) - \frac{2}{|n|} \sum_{\{i|y_i=-1\}} \kappa(x, x_i) - \\ - \frac{1}{|p|^2} \sum_{\{i|y_i=+1\}} \sum_{\{j|y_j=+1\}} \kappa(x_i, x_j) + \frac{1}{|n|^2} \sum_{\{i|y_i=-1\}} \sum_{\{j|y_j=-1\}} \kappa(x_i, x_j) & \end{aligned}$$

* En anglès, *kernel matrix*.

Matriu semidefinida positiva

Es diu que una matriu simètrica és semidefinida positiva quan tots els seus valors propis són superiors o iguals a zero.

I d'això obtenim:

$$b = \frac{1}{2} \left(\frac{1}{|p|^2} \sum_{\{i|y_i=+1\}} \sum_{\{j|y_j=+1\}} \kappa(x_i, x_j) - \frac{1}{|n|^2} \sum_{\{i|y_i=-1\}} \sum_{\{j|y_j=-1\}} \kappa(x_i, x_j) \right)$$

$$h(x) = \text{signe} \left(\frac{1}{|p|} \sum_{\{i|y_i=+1\}} \kappa(x, x_i) - \frac{1}{|n|} \sum_{\{i|y_i=-1\}} \kappa(x, x_i) - b \right)$$

Noteu que en aquestes expressions resultants no apareix ϕ per cap costat. Aquesta forma en què no apareix ϕ rep el nom de *formulació dual*, en contraposició a la *formulació primal*. Una altra qüestió que cal tenir en compte de cara a la implementació és que la b és una constant que s'ha de calcular una sola vegada, ja que no hi apareix l'exemple de test. No l'haurem de recalcular per a cada un dels exemples de test.

Construcció de *kernels*

Hi ha un conjunt de regles anomenades *propietats de la clausura* que ens ajuden en la construcció de nous *kernels*. Si tenim que κ_1 i κ_2 són *kernels* sobre $X \times X$, $X \subseteq \mathbb{R}^n$, $a \in \mathbb{R}^+$, $f(\cdot)$ una funció real sobre X , $\phi : X \rightarrow \mathbb{R}^N$ amb κ_3 un *kernel* sobre $\mathbb{R}^N \times \mathbb{R}^N$, i B una matriu simètrica semidefinida positiva de $n \times n$, les funcions següents són *kernels*:

- 1) $\kappa(x, z) = \kappa_1(x, z) + \kappa_2(x, z)$
- 2) $\kappa(x, z) = a\kappa_1(x, z)$
- 3) $\kappa(x, z) = \kappa_1(x, z)\kappa_2(x, z)$
- 4) $\kappa(x, z) = f(x)f(z)$
- 5) $\kappa(x, z) = \kappa_3(\phi(x), \phi(z))$
- 6) $\kappa(x, z) = x'Bz$

Kernels habituals

Hi ha dos *kernels* no lineals d'aplicació general que apareixen en totes les implementacions de mètodes basats en *kernels*. A continuació en donem la caracterització.

Si $\kappa_1(x, z)$ és un *kernel* sobre $X \times X$, en què $x, z \in X$, i $p(x)$ és un polinomi amb coeficients positius i $\sigma > 0$, llavors les funcions següents també són *kernels*:

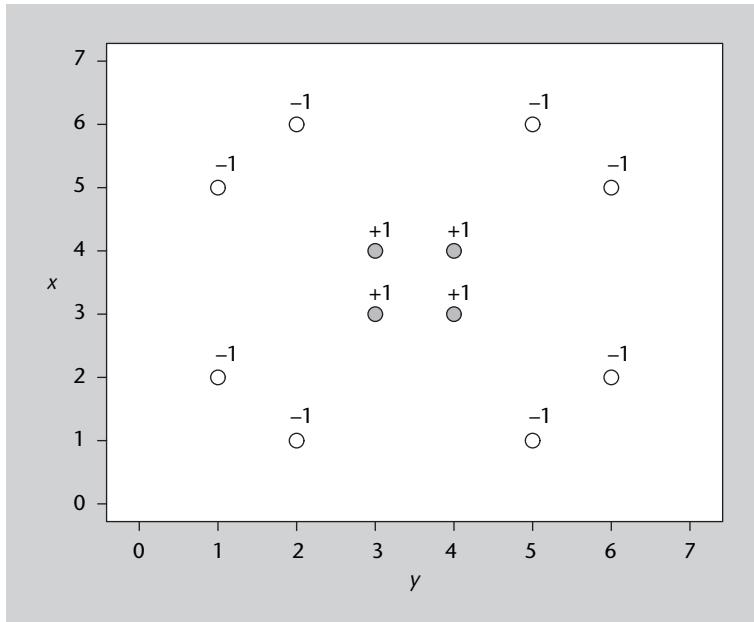
- 1) $\kappa(x, z) = p(\kappa_1(x, z))$
- 2) $\kappa(x, z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$

El primer *kernel* és conegut com a *kernel polinòmic* i el segon com a *kernel gaussià* o *RBF kernel*.

Exemple d'aplicació

El conjunt de dades que mostra la figura 32 no és linealment separable. Si calculem els centroides i apliquem el conjunt d'entrenament com a conjunt de test, aconseguim una precisió del 66%. Com podem separar aquest conjunt?

Figura 32. Exemple de conjunt linealment no separable



Aquest conjunt de dades es pot aprendre amb un *kernel* radial

$$\kappa(x, z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$$

que de vegades apareix com a $\kappa(x, z) = \exp(-\gamma\|x-z\|^2)$.

Si escollim com a conjunt d'entrenament els punts de la figura 32, com a exemple de test el punt $x_t = (3, 6)$ i un *kernel* radial amb $\gamma = 1$, obtenim:

$$b = 0,1629205972066024$$

$$h(x_t) = \text{signe}(-0,2057373019348639) = -1$$

La predicció de -1 és coherent amb la posició del punt $(3, 6)$ en la figura 32.

Implementació en Python

El programa 4.8 correspon a la implementació en Python del classificador lineal binari basat en els centroides amb *kernel* gaussià. Aquest programa funcionarà per a tots els problemes amb dades numèriques sense valors absents que tinguin la classe com a última columna i codificada com a -1 i +1.

Codi 4.8: Lineal amb *kernel* RBF en Python

```

1  from functools import reduce
2  from math import exp
3
4  # declaració de kernel
5
6  def kernel(a, b, gamma=1):
7      return exp(-gamma *
8          sum(map(lambda x, y:
9              (float(x) - float(y)) ** 2, a, b)))
10
11 # declaració de funcions
12
13 def h(x, tp, tn, cp, cn, b):
14     y = (sum([kernel(x, xi) for xi in tp]) / cp -
15         sum([kernel(x, xi) for xi in tn]) / cn - b)
16     if y > 0:
17         return '+1'
18     else:
19         return '-1'
20
21 train = list(map(lambda l: (l.strip()).split(','), 
22                  open('punts.txt', 'r').readlines()))
23
24 # Entrenament
25 tp = list(*filter(lambda x: x[len(train[0])-1] == '+1', train))
26 tp.pop(len(tp)-1)
27 tp = list(zip(*tp))
28 cp = len(tp)
29 tn = list(*filter(lambda x: x[len(train[0])-1] == '-1', train))
30 tn.pop(len(tn)-1)
31 tn = list(zip(*tn))
32 cn = len(tn)
33
34 b = (sum([sum([kernel(xi, xj) for xi in tp])
35           for xj in tp]) / (cp ** 2) -
36       sum([sum([kernel(xi, xj) for xi in tn])
37             for xj in tn]) / (cn ** 2)) / 2
38
39 test = [[3, 6]]
40 classesTest = [-1]
41 prediccions = [h(x, tp, tn, cp, cn, b) for x in test]
42
43 ### Nombre de correctes
44 print('Prec.:', len(list(filter(lambda x: x[0] == x[1],
45                                zip(*[prediccions, classesTest])))) /
46       len(test) * 100, '%')

```

Si apliquem el programa a l'arxiu de dades del problema de la figura 32 obtindrem una predicció correcta per al punt (3,6).

Anàlisi del mètode

La definició de *kernels* està a mig camí entre la representació de la informació i l'algorisme d'aprenentatge. Els *kernels* estan molt relacionats amb l'espai d'a-

Repositori

Els arxius de dades i el programa estan disponibles en el repositori de l'assignatura.

atributs dels problemes. És a dir, no podem dir que hi hagi un *kernel* millor que un altre. Cada *kernel* funciona amb els problemes als quals millor s'adapta per les seves característiques intrínseqües. Per al problema de les flors que mostra la taula 27, funciona bé el *kernel* lineal però no funciona el *kernel* radial, i per al que mostra la figura 32, passa exactament el contrari.

L'ús de *kernels* ens permet adaptar els algorismes a espais d'atributs més complexos o específics al problema. Amb aquests s'ha aconseguit millorar el comportament dels algorismes d'aprenentatge per a molts problemes, abordar problemes que fins al moment no tenien solució i anar més enllà i crear nous algorismes que tracten dades estructurades, com poden ser les anàlisis sintàctiques del llenguatge natural o el tractament de vídeos. A més, l'accés a aquests espais d'atributs complexos i flexibles s'aconsegueix amb un cost computacional (en espai i temps) relativament baix.

Un dels grans avantatges d'aquests mètodes, a més de la seva flexibilitat, és que estan teòricament molt ben fonamentats i s'han demostrat característiques com l'eficiència computacional, robustesa i estabilitat estadística.

En la bibliografia es parla del coll d'ampolla de la matriu *kernel*, com del límit que imposa treballar-hi per a tot. Tot el que vulguem representar dels problemes s'ha de poder expressar en forma de funcions *kernel* i, per tant, de la matriu *kernel*. En alguns casos podem trobar limitacions.

L'ús de *kernels* no és exclusiu dels algorismes de classificació. Aquesta mateixa tècnica s'ha utilitzat en mètodes de *clustering* com els que es descriuen en el subapartat 2.3. o en els algorismes de selecció d'atributs descrits en l'apartat 3. Així, hi ha algorismes com el *kernel PCA* o el *kernel k-means*. Els algorismes com el *kernel PCA* o el *kernel CCA* s'estan utilitzant en problemes de visió com el reconeixement de cares.

Una dels principals avantatges de l'ús de *kernels* (funcions nucli) és el disseny de *kernels* específics per a tasques concretes. En els últims temps s'està investigant molt en la creació de *kernels* per a tractar conjunts, cadenes de text, grafs o arbres, entre altres*. Alguns d'aquests *kernels* són els específics per a tractar text**, imatges o vídeos***.

* J. Shawe-Taylor; N. Cristianini (2004). *Kernel Methods for Pattern Analysis*. RU: Cambridge University Press.

** T. Joachims (2001). *Learning to Classify Text Using Support Vector Machines*. Kluwer Academic Publishers.

*** F. Camastra; A. Vinciarelli (2008). "Machine Learning for Audio, Image and Video Analysis" (Advanced Information and Knowledge Processing). Springer.

4.5.3. Kernels per a tractament de textos

En aquest subapartat aprofundirem en un dels *kernels* específics, el *kernel* per al tractament de textos, que ens servirà per a abordar el problema de la categorització de textos.

Suposem que volem dissenyar un classificador per a una agència de notícies que sàpiga distingir els documents relacionats amb *adquisicions corporatives*.

El primer pas que hem de seguir per a poder abordar aquest problema és pensar en la codificació de les dades. Una de les maneres més simples de representar text és mitjançant un conjunt de paraules*. A cada paraula se li pot assignar si apareix o no en el document, el nombre de vegades que apareix o la freqüència relativa de la paraula en el document. La freqüència relativa evita el biaix que produeix la mida dels exemples (documents) en els resultats.

* En anglès, *bag of words*.

En el conjunt de dades de què disposem*, ja està preprocesat i el vector conté la informació de les freqüències relatives. La taula 29 mostra la primera part del contingut del primer exemple d'entrenament. El primer valor correspon a la classe, que pot ser positiva o negativa. En aquest cas està indicant que l'exemple pertany a les *adquisicions corporatives*. Les columnes següents corresponen a parelles *atribut : valor*. Així 6 : 0,0198 ens diu que l'atribut número 6 té el valor 0,0198. Els atributs estan ordenats i aquells que no apareixen vol dir que són valors nuls. Aquest tipus de representació dels atributs es denomina representació dispersa**, en contraposició de la representació densa. En un arxiu separat trobem un índex de paraules en què podem comprovar que l'atribut 6 correspon a l'aparició de la paraula *said*. Els conjunts d'entrenament i test estan separats en dos arxius: *train.dat* i *test.dat*.

* Aquest conjunt de dades és un subconjunt del Reuters 21578 processat per T. Joachims que està disponible en:
<http://svmlight.joachims.org>

** En anglès, *sparse*.

Taula 29. Exemple de representació d'un document

```
+1 6:0.0198403253586671 15:0.0339873732306071 29:0.0360280968798065 ...
```

Font: problema de "text categorisation" formatat per T. Joachims.

A partir d'això, podem definir que la representació d'un document d queda de la manera:

$$\phi(d) = (tf(t_1, d), tf(t_2, d), \dots, tf(t_N, d)) \in \mathbb{R}^N$$

en què t_i correspon al terme o paraula i , N al nombre de termes i $tf(t_i, d)$ a la freqüència relativa del terme t_i en el document d .

Partint d'aquesta definició, el conjunt d'entrenament el podem expressar mitjançant la matriu següent:

$$D = \begin{pmatrix} tf(t_1, d_1) & tf(t_2, d_1) & \cdots & tf(t_N, d_1) \\ tf(t_1, d_2) & tf(t_2, d_2) & \cdots & tf(t_N, d_2) \\ \vdots & \vdots & \ddots & \vdots \\ tf(t_1, d_l) & tf(t_2, d_l) & \cdots & tf(t_N, d_l) \end{pmatrix}$$

en què cada fila correspon a un document i cada columna a l'aparició d'un terme en tots els documents.

A partir d'això es poden definir diferents tipus de funcions *kernel*. La primera aproximació a una funció de *kernel* és una de les que s'usa per a tractar conjunts:

$$\kappa(d_1, d_2) = 2^{|A_1 \cap A_2|}$$

en què A_i correspon al conjunt format per les paraules de d_i .

En categorització de textos se sol utilitzar la composició de *kernels* per a anar creant *kernels* més sofisticats. Per a veure'n un exemple, anem a veure com es combinaria el *kernel* anterior amb l'RBF estudiat anteriorment.

Si partim dels *kernels*:

$$\kappa_2(d_1, d_2) = 2^{|A_1 \cap A_2|}$$

$$\kappa_1(x, z) = \exp(-\gamma \|x, z\|)$$

i que

$$\|\phi_1(x) - \phi_1(z)\|^2 = \kappa_1(x, x) - 2\kappa_1(x, z) + \kappa_1(z, z)$$

el *kernel* resultat de la combinació és:

$$\kappa(d_1, d_2) = \exp(-\gamma(\kappa_2(d_1, d_1) - 2\kappa_2(d_1, d_2) + \kappa_2(d_2, d_2)))$$

Una segona aproximació per a abordar la creació de *kernels* per a categorització de textos més utilitzada que la de conjunts és la “kernelització” del VSM*. Aquest *kernel*, a partir de la definició anterior de la matriu D , es formula com segueix:

* De l'anglès, *vector space model*.
El *kernel* associat rep el nom de *vector space kernel*.

$$\kappa(d_1, d_2) = \sum_{i=1}^N tf(t_i, d_1)tf(t_i, d_2)$$

Implementació en Python

El programa 4.9 correspon a la implementació en Python del classificador lineal binari basat en els centroides amb els tres *kernels* anteriors. Aquest programa funcionarà per a tots els problemes amb el format descrit en el subapartat anterior.

Codi 4.9: Lineal amb *kernel RBF* en Python

```

1  from functools import reduce
2  from math import exp
3  from time import clock
4
5  # declaració de kernel
6
7  def kernelSet(a, b):
8      da = dict((k,v) for k, v in (l.split(':') for l in a))
9      db = dict((k,v) for k, v in (l.split(':') for l in b))
10     return 2 ** sum(min(float(da[key]), float(db[key])))
11             for key in da.keys() if key in db)
12
13 def kernelSetRBF(a, b, gamma=1):
14     return exp(-gamma * (kernelSet(a, a) - 2 *
15                         kernelSet(a, b) + kernelSet(b, b)))
16
17 def kernelVSM(a, b):
18     da = dict((k,v) for k, v in (l.split(':') for l in a))
19     db = dict((k,v) for k, v in (l.split(':') for l in b))
20     return 2 ** sum(float(da[key]) * float(db[key]))
21             for key in da.keys() if key in db)
22
23 kernel = kernelVSM
24
25 # declaració de funcions
26
27 def h(x, tp, tn, cp, cn, b):
28     y = (sum([kernel(x, xi) for xi in tp]) / cp -
29           sum([kernel(x, xi) for xi in tn]) / cn - b)
30     if y > 0:
31         return '+1'
32     else:
33         return '-1'
34
35 # càrrega de l'arxiu
36 train = list(map(lambda l: (l.strip()).split('_'),
37                  filter(lambda x: x[0] != '#',
38                        open('train.dat', 'r').readlines())))
39
40 # Entrenament
41 tp = list(filter(lambda x: x[0] == '1', train))
42 l= list(map(lambda x: x.pop(0), tp))
43 del(l)
44 cp = len(tp)
45
46 tn = list(filter(lambda x: x[0] == '-1', train))
47 l= list(map(lambda x: x.pop(0), tn))
48 del(l)
49 cn = len(tn)
50
51 del(train)
52
53 inici = clock()
54 b = (sum([sum([kernel(xi, xj) for xi in tp])
55           for xj in tp]) / (cp ** 2) -
56           sum([sum([kernel(xi, xj) for xi in tn])
57                 for xj in tn]) / (cn ** 2)) / 2
58 print("b", b, clock() - inici, "s")
59
60 test = list(map(lambda l: (l.strip()).split('_'),
61                  filter(lambda x: x[0] != '#',
62                        open('test.dat', 'r').readlines())))
63 classesTest=list(map(lambda x: x.pop(0), test))
64
65 inici = clock()
66 prediccions = [h(x, tp, tn, cp, cn, b) for x in test]
67 print("h", clock() - inici, "s")
68
69 ### Nombre de correctes
70 print('Prec.:', len(list(filter(lambda x: x[0] == x[1],
71                               zip(*[prediccions, classesTest]))))
72     / len(test) * 100, '%')

```

La taula 30 mostra els resultats en aplicar el programa 4.9 al conjunt de dades anterior amb els tres tipus de *kernel** i el temps de càlcul del paràmetre *b* i les prediccions *h*. El programa s'ha executat en un portàtil Mac amb OS X d'1,4 GHz i 2 GB de memòria.

* Fixeu-vos que per a seleccionar el *kernel* hi ha una línia just sota les seves definicions de l'estil:
`kernel = kernelSet`

Taula 30. Taula de resultats per al primer problema de text categorització

kernel	precisió	temps <i>b</i>	temps <i>h</i>
conjunt	84%	452 s	246 s
conjunt + RBF	51,2%	1805 s	998 s
VSM	88%	440 s	244 s

Repositori

Els arxius de dades i el programa estan disponibles en el repositori de l'assignatura.

Suposem que volem dissenyar un classificador per a distingir pàgines web de dues temàtiques diferents. Per a aquest problema utilitzarem com a temes dos de les etiquetes de l'ontologia ODP: *Arts/Music/Bands_and_Artists/U* i *Arts/Celebrities/S*.

En aquest cas, disposem d'un únic arxiu processat, en el qual està desat el nombre de vegades que apareix cada paraula en el document. El format és el mateix que per al problema anterior.

Implementació en Python

El programa 4.10 correspon a la implementació en Python del classificador lineal binari basat en els centroides amb el kernel VSM i un que el combina amb l'RBF. En aquest cas, hem modificat el *kernel* VSM, per raons d'eficiència, respecte al que apareix en el codi 4.9.

Codi 4.10: Lineal amb *kernel* RBF en Python

```

1 from functools import reduce
2 from math import exp
3 from time import clock
4
5 # declaració de kernel
6
7 def kernelVSM(a, b):
8     la = [(int(k),int(v)) for k, v in (l.split(':') for l in a)]
9     lb = [(int(k),int(v)) for k, v in (l.split(':') for l in b)]
10    s = 0
11    i = 0
12    j = 0
13    while (i < len(la)) and (j < len(lb)):
14        if la[i][0] == lb[j][0]:
15            s += la[i][1] * lb[j][1]
16            i += 1
17            j += 1
18        elif la[i][0] < lb[j][0]:
19            i += 1
20        else:
21            j += 1
22    return s
23
24 def kernelVSMRBF(a, b, gamma=1):
25     return exp(-gamma * (kernelVSM(a, a) - 2 *
26                         kernelVSM(a, b) + kernelVSM(b, b)))
27
28 kernel = kernelVSMRBF

```

```

29
30 # declaració de funcions
31
32 def h(x, tp, tn, cp, cn, b):
33     y = (sum([kernel(x, xi) for xi in tp]) / cp -
34           sum([kernel(x, xi) for xi in tn]) / cn - b)
35     if y > 0:
36         return '+1'
37     else:
38         return '-1'
39
40 # càrrega de l'arxiu
41 data = list(map(lambda l: (l.strip()).split('_'),
42                 filter(lambda x: x[0] != '#',
43                        open('vectors.dat', 'r').readlines())))
44
45 # construcció del train: 2 de cada 3
46 train = list(map(lambda x: x[1],
47                  filter(lambda v: v[0] % 3 != 0, enumerate(data))))
48
49 # construcció del test: 1 de cada 3
50 test = list(map(lambda x: x[1],
51                 filter(lambda v: v[0] % 3 == 0, enumerate(data))))
52 classesTest = list(map(lambda x: x.pop(0), test))
53
54 del(data)
55
56 # Entrenament
57 tp = list(filter(lambda x: x[0] == '+1', train))
58 l= list(map(lambda x: x.pop(0), tp))
59 del(l)
60 cp = len(tp)
61
62 tn = list(filter(lambda x: x[0] == '-1', train))
63 l= list(map(lambda x: x.pop(0), tn))
64 del(l)
65 cn = len(tn)
66
67 del(train)
68
69 inici = clock()
70 b = (sum([sum([kernel(xi, xj) for xi in tp])
71           for xj in tp]) / float(cp ** 2) -
72       sum([sum([kernel(xi, xj) for xi in tn])
73             for xj in tn]) / float(cn ** 2)) / 2.0
74 print("b", b, clock() - inici, "s")
75
76 inici = clock()
77 prediccions = [h(x, tp, tn, cp, cn, b) for x in test]
78 print("h", clock() - inici, "s")
79
80 ### Nombre de correctes
81 print('Prec.: ', len(list(filter(lambda x: x[0] == x[1],
82                               zip(*[prediccions, classesTest])))) /
83     len(test) * 100, '%')

```

La taula 31 mostra els resultats en aplicar el programa 4.10 al conjunt de dades anterior amb els dos tipus de *kernel** i el temps de còmput del paràmetre *b* i les prediccions. El programa s'ha executat en un portàtil Mac amb OS X d'1,4 GHz i 2 GB de memòria.

Taula 31. Taula de resultats per al primer problema de text categorització

kernel	precisió	temps <i>b</i>	temps <i>h</i>
VSM	61,8%	55 s	48 s
VSM + RBF	52,7%	164 s	145 s

* Fixeu-vos que per a seleccionar el *kernel* hi ha una línia just sota les definicions de l'estil:
kernel = *kernelVSM*

Repositori

Els arxius de dades i el programa estan disponibles en el repositori de l'assignatura.

Ampliacions de *kernels* per a text

Quan es treballa en classificació de textos se solen preprocessar els documents. Un dels processos més comuns és eliminar les paraules sense contingut semàntic com: preposicions, articles... Hi ha llistes d'aquestes paraules que ens ajuden a fer aquesta tasca.

El segon procés que se sol utilitzar és eliminar del diccionari aquelles paraules que no apareixen més de dues o tres vegades en el conjunt de tots els documents del corpus.

Amb això aconseguim reduir bastant l'espai d'atributs i, així, millorar l'eficiència computacional dels algorismes esperant no degradar-ne la precisió.

En el segon exemple de classificació de textos que hem vist no hem tingut en compte el tractament de la mida dels documents. Això esbiaixarà el comportament de les prediccions envers els documents més grans, ja que contenen més paraules i tenen més probabilitats d'enllaçar-se entre ells. Una altra manera de tractar aquest tema, a part de codificar freqüències com en el primer problema tractat, és normalitzar el *kernel* resultant. Això es fa com a etapa final de la combinació de *kernels* a partir de la formulació següent:

$$\hat{k}(x,y) = \left\langle \frac{\phi(x)}{\|\phi(x)\|}, \frac{\phi(y)}{\|\phi(y)\|} \right\rangle = \frac{k(x,y)}{\sqrt{k(x,x)k(y,y)}}$$

Com a punt final d'aquest subapartat descriurem una funció *kernel* basada en un dels algorismes més utilitzats de la recuperació de la informació, el *tf-idf*. La idea que persegueix aquest algorisme és doble: d'una banda aporta una ponderació dels termes o paraules en funció de la seva importància; i d'una altra, aporta mesures de relació entre els termes, que ens ajuden a relacionar documents que no comparteixen exactament els mateixos termes però sí sónònims.

Anem a començar representant la definició de funció *kernel* del VSM:

$$k(d_1, d_2) = \langle \phi(d_1), \phi(d_2) \rangle = \sum_{i=1}^N tf(t_i, d_1)tf(t_i, d_2)$$

Si volem ponderar els termes amb un pes, necessitem una funció $w(t)$ que ens ho proporcioni. Hi ha diferents alternatives, com poden ser la informació mútua o mesures basades en l'entropia. El *tf-idf* utilitza la funció:

$$w(t) = \ln \left(\frac{l}{df(t)} \right)$$

Repositori

Teniu disponible en el repositori de l'assignatura llistes de paraules sense contingut semàntic per a anglès, castellà i català.

Diccionari

Es coneix com a diccionari l'arxiu índex de paraules que apareixen en els documents del conjunt de dades.

Vegeu també

Per a saber més sobre la informació mútua podeu veure el subapartat 3.1.3. Per a saber més sobre mesures basades en l'entropia podeu veure el subapartat 4.4.1.

en què l correspon al nombre total de documents i $df(t)$ al nombre de documents que contenen el terme t .

Si generem un *kernel* a partir d'aquesta funció obtindrem:

$$\tilde{\kappa}(d_1, d_2) = \sum_t w(t)^2 tf(t, d_1) tf(t, d_2)$$

Si volem afegir semblances semàntiques entre termes, hi ha diverses maneres de fer-ho. Una és explotar una ontologia de lliure distribució molt utilitzada en el camp del processament del llenguatge natural que es denomina *WordNet**. Aquesta ontologia conté les diferents paraules de la llengua organitzades en una jerarquia en la qual cada node representa un concepte i conté totes les paraules sinònimes que el representen. Una paraula està en els diferents nodes als quals pertany en funció dels seus diferents significats. Els nodes estan interconnectats per mitjà de la relació d'hiperonímia/hiponímia**. Partint d'aquesta ontologia, la semblança entre paraules té a veure amb la inversa del camí més curt per a arribar d'una paraula a l'altra.

*<http://wordnet.princeton.edu>

** A manera d'exemple: *felí* és un hiperònim de *gat* i *gat* és hipònim de *felí*.

En recuperació d'informació es juga amb la matriu D per a obtenir un resultat semblant. Recordem que la matriu de *kernel* es pot generar a partir de:

$$K = D \times D'$$

i que:

$$D = \begin{pmatrix} tf(t_1, d_1) & tf(t_2, d_1) & \cdots & tf(t_N, d_1) \\ tf(t_1, d_2) & tf(t_2, d_2) & \cdots & tf(t_N, d_2) \\ \vdots & \vdots & \ddots & \vdots \\ tf(t_1, d_l) & tf(t_2, d_l) & \cdots & tf(t_N, d_l) \end{pmatrix}$$

Definim que dos termes estan relacionats entre si si apareixen freqüentment junts en els mateixos documents. Això ho podem aconseguir a partir de:

$$D' \times D$$

Fixeu-vos que la matriu $D'D$ tindrà un valor diferent de zero en la posició (i,j) si, i només si, hi ha algun document en el corpus en el qual coocorren els termes t_i i t_j .

A partir d'això, podem definir un nou *kernel*:

$$\tilde{\kappa}(d_1, d_2) = \phi(d_1) D' D \phi(d_2)'$$

en què

$$(D'D)_{i,j} = \sum_d tf(i,d)tf(j,d)$$

L'ús d'aquestes semblances semàntiques es coneix com a *GVSM*, en què la *G* ve de *generalitzat*.

4.5.4. Màquines de vectors de suport

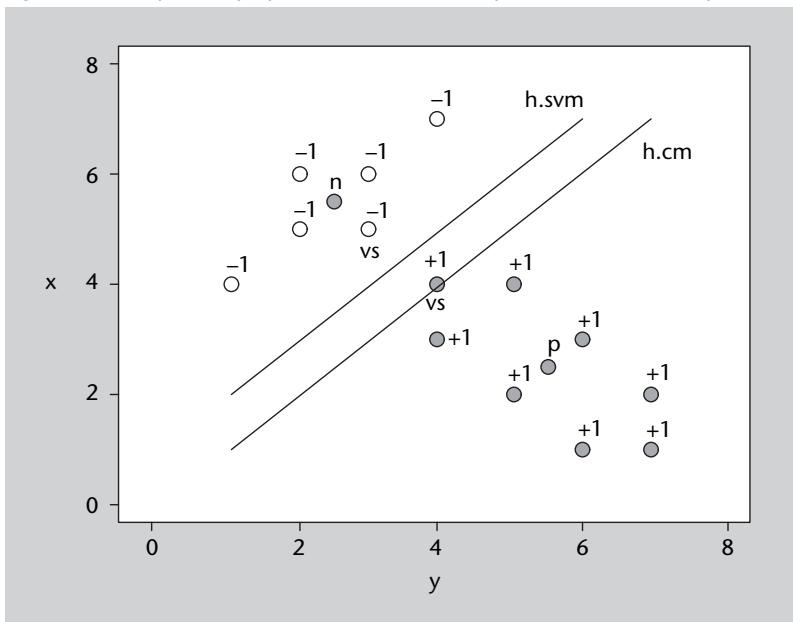
Suposem que volem dissenyar un classificador per a una agència de notícies que sàpiga distingir els documents relacionats amb *adquisicions corporatives*.

Les màquines de vectors de suport* (SVM) són un algorisme que millora el classificador lineal buscant un hiperplà millor que el que es genera amb el classificador lineal.

* En anglès, *support vector machines*: N. Cristianini; J. Shawe-Taylor (2000). *An Introduction to Support Vector Machines (SVMs)*. Cambridge University Press.

La figura 33 il·lustra aquest concepte: els punts marcats amb +1 corresponen als exemples positius, els -1 als negatius, *p* al centroïde dels positius, *n* al dels negatius i *h.cm* a l'hiperplà generat pels centroides (està situat a mig camí dels dos centroides).

Figura 33. Exemples d'hiperplans lineals i de les màquines de vectors de suport



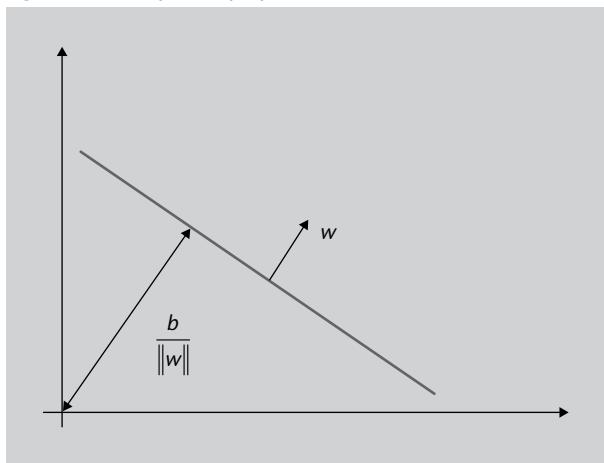
Les màquines de vectors de suport són un algorisme d'optimització (com els descrits en l'apartat 5) que escull l'hiperplà amb marge màxim entre tots els possibles hiperplans que separen els exemples positius dels negatius (el que té la mateixa distància als exemples positius que als negatius). Descriu l'hiperplà a partir dels anomenats *vectors de suport*. Aquests solen ser els punts més propers a l'hiperplà i els que el defineixen. En la figura 33, $h.svm$ correspon a l'hiperplà de marge màxim que trobarien les SVM per a aquest conjunt i els vectors de suport estan marcats amb vs .

Interpretació geomètrica

Aquest algorisme combina la maximització del marge, com l'AdaBoost, amb l'ús dels *kernels* descrits en el subapartat anterior i la possibilitat de flexibilitzar el marge. Aquest subapartat pretén exposar aquests conceptes de manera visual.

Molts mètodes d'aprenentatge es basen en la construcció d'*hiperplans*. Un hiperplà és una figura geomètrica que té una dimensió menys que l'espai en el qual està situat. Exemples d'hiperplans són línies en \mathbb{R}^2 o plans en \mathbb{R}^3 . La figura 34 ens mostra un exemple d'hiperplà en \mathbb{R}^2 . Un hiperplà queda definit per un vector de pesos (w) i un llindar (b).

Figura 34. Exemple d'hiperplà



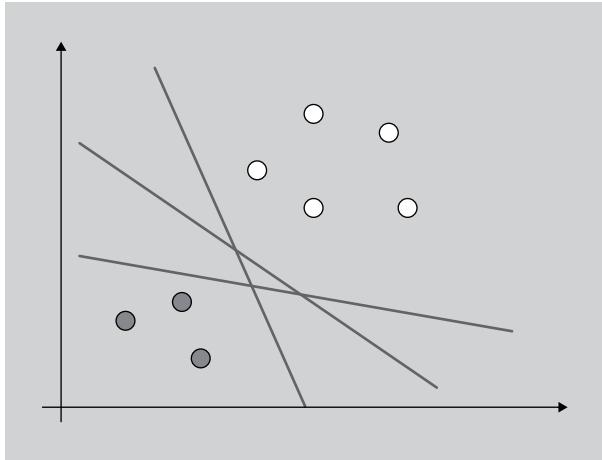
Considerem ara un exemple en \mathbb{R}^2 . Hem de construir un classificador per a separar dos tipus de punts. La figura 35 mostra tres hiperplans vàlids (en aquest cas línies) que ens separarien els dos conjunts de punts.

Podríem trobar infinitis hiperplans per a aquest problema de classificació. Una vegada hem escollit un hiperplà per a classificar un conjunt de punts, podem crear una regla de classificació a partir d'aquest. Aquesta regla queda definida per l'equació:

$$h(x) = \text{signe} \left(b + \sum_{i=1}^N w_i \cdot x_i \right)$$

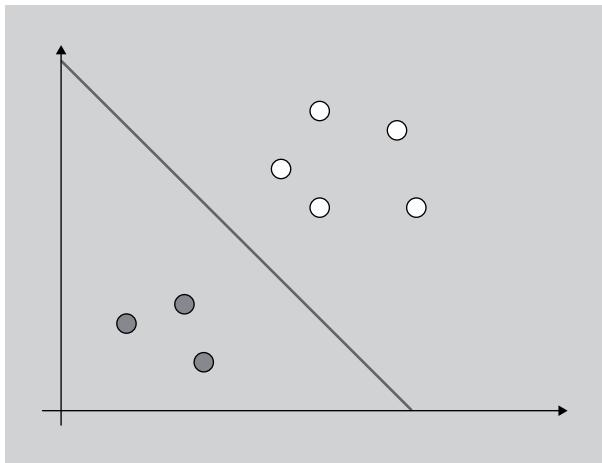
en què N és el nombre d'atributs i el vector $w = \{w_1, \dots, w_N\}$ i b defineixen l'hiperplà i $x = \{x_1, \dots, x_N\}$ l'exemple per classificar.

Figura 35. Exemple d'hiperplans possibles



Però quin d'aquests infinitis hiperplans ens servirà millor com a classificador? Intuïtivament es pot dir que la millor recta per a dividir aquests conjunts seria semblant a la de la figura 36.

Figura 36. Millor hiperplà

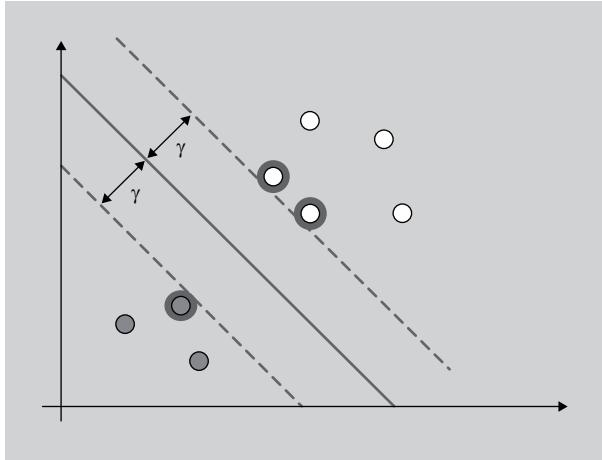


Per què? Perquè és la que deixa més distància a tots els punts; és la recta que passa més allunyada dels punts de les dues classes. Aquest concepte es denomina *marge*. La figura 37 ens mostra la representació geomètrica d'aquest concepte; en què γ correspon al marge i els punts marcats en blau als vectors de suport. Els vectors de suport són aquells punts que estan fregant el marge.

Les màquines de vectors de suport són un mètode d'aprenentatge que es basa en la maximització del marge. Com ho fa? L'estrategia del mètode consisteix a buscar els vectors de suport. Aquest mecanisme de cerca parteix d'un espai d'hipòtesi de funcions lineals en un espai d'atributs altament multidimensional. S'hi entrena amb un algorisme de la teoria de l'optimització derivat de la

teoria de l'aprenentatge estadístic. El subapartat següent descriu aquest procés formalment. Aquest apartat està orientat a veure'n les aplicacions pràctiques.

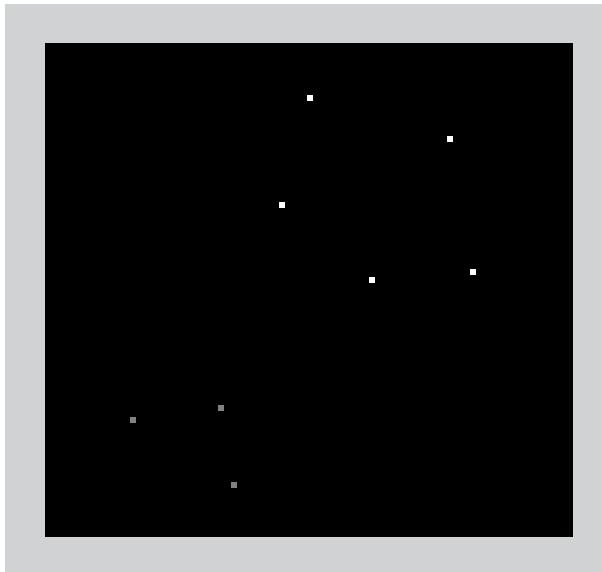
Figura 37. Marge



A partir d'aquest moment utilitzarem el programari *svmtoy* per a il·lustrar tots els conceptes. Aquest programa aplica les SVM amb diferents paràmetres a un conjunt amb dos atributs i representa gràficament el resultat de la classificació.

Començarem amb el cas anterior. Introduirem una distribució semblant a la de la figura 37 (com mostra la figura 38). Els punts s'introdueixen amb el botó del ratolí. Cal prémer el botó *change* per a canviar de classe. Una vegada introduïts els punts, cal especificar els paràmetres. Per a entrenar aquest cas especificarem: $-t 0 -c 100$. És a dir, un *kernel* lineal ($-t 0$) i un valor de 100 a la rigidesa (inversa de la flexibilitat) del marge. Aquest últim concepte el veurem més endavant.

Figura 38. Configuració inicial per a l'svmtoy



svmtoy

svmtoy és un programa inclòs en la *libsvm*: Chih-Chung Chang; Chih-Jen Lin (2001). LIBSVM: a library for support vector machines.
<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

Repositori

El programari *svmtoy* està disponible en el repositori de l'assignatura.

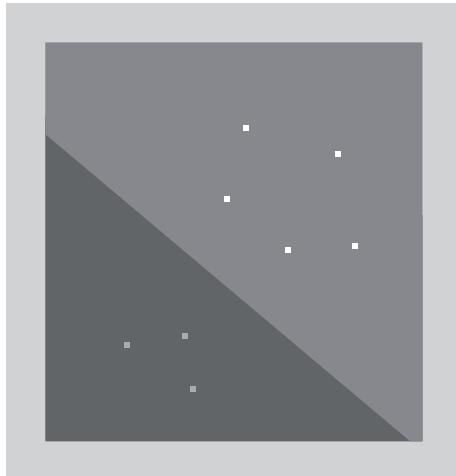
Kernel lineal

Un *kernel* lineal és de l'estil $u' \times v$ en què u' és el nostre vector d'atributs i v és un altre vector.

Una vegada establerts els paràmetres, premem en *run* i ens apareixerà una pantalla com la que mostra la figura 39. Podem observar com ha separat bé els

dos conjunts de punts. Es pot intuir un marge clar i que els vectors de suport són dos punts grocs i un blau (els més propers a l'hiperplà).

Figura 39. Kernel lineal amb *svmttoy*

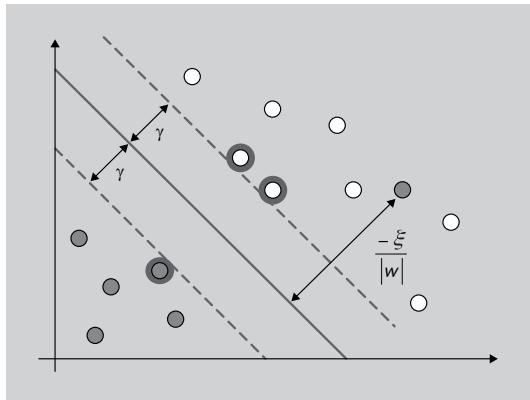


Quan donat un problema de punts podem trobar un hiperplà lineal (en el cas de \mathbb{R}^2 , una recta) que divideix perfectament els dos conjunts de punts, tenim un problema, que denominem *linealment separable*. El cas anterior compleix aquesta propietat. Pot ocurrir (més aviat, sol ocurrir) que els conjunts de dades no siguin tan clarament separables. En aquests casos ens solem trobar amb dues tipologies de problemes:

- El primer problema apareix quan hi ha punts solts de les dues classes a la zona del marge, o punts d'una classe a la zona de l'altra. El cas mostrat en el figura 40 no és linealment separable. El punt vermell que hi ha entre els verds normalment és un error o un *outlier*. La solució implica generar un *marge flexible**. El marge flexible admet errors en la separació en canvi d'augmentar el marge. Aquests errors estan ponderats per l'expressió $-\xi/|w|$. L'ús de marges flexibles normalment augmenta la generalització i disminueix la variància del model. El paràmetre $-c$ de l'*svmttoy* codifica la rigidesa del marge. Com més petit sigui el valor d'aquest paràmetre, més flexible és el marge. O dit d'una altra manera, la flexibilitat del marge és inversament proporcional al valor del paràmetre c . Aquest paràmetre s'escull normalment com una potència de deu ($10^{\pm x}$).

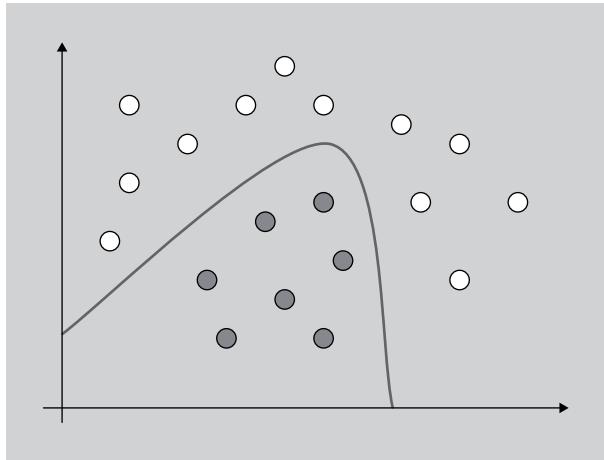
* En anglès, *soft margin*.

Figura 40. Marge flexible



- La segona tipologia de problemes apareix quan els punts no tenen una distribució que es pugui separar linealment. És a dir, l'estructura del problema segueix un altre tipus de distribucions. Per a un problema com el que mostra la figura 41 ens aniria bé una corba en lloc d'una recta per a separar els conjunts de punts. Això s'aconsegueix canviant de *kernel*. Un *kernel* és una funció de projecció entre espais. Aquest mecanisme ens permet augmentar la complexitat del model, i per tant, la variància.

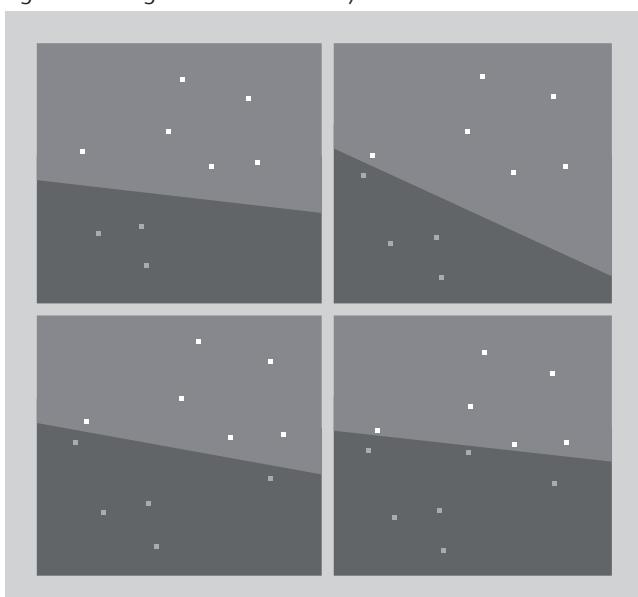
Figura 41. Conjunt linealment no separable



Amb vista a construir models amb les SVM, és molt interessant estudiar el comportament d'aquests dos paràmetres: els *kernels* i els marges flexibles.

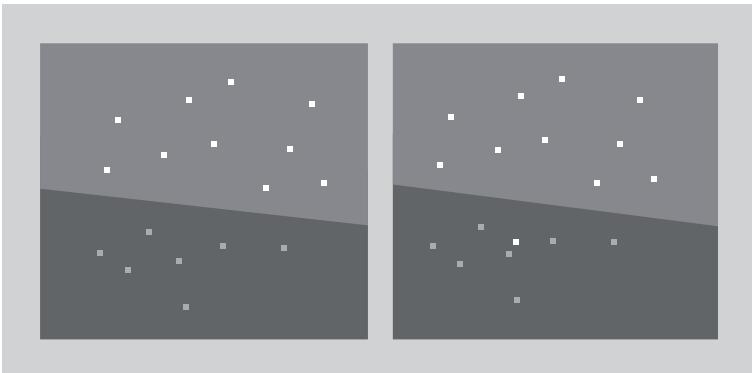
Agafem ara el conjunt de la figura 39 i comencem a afegir-hi punts. Fixeu-vos com van canviant l'hiperplà i el marge en les imatges de la figura 42 (haurem d'augmentar la rigidesa de l'hiperplà escollint un valor de 10.000 per al paràmetre c). En l'última imatge de la figura 42, la distància dels vectors de suport a l'hiperplà és molt inferior a la de la primera.

Figura 42. Marge flexible amb *svmtoy*



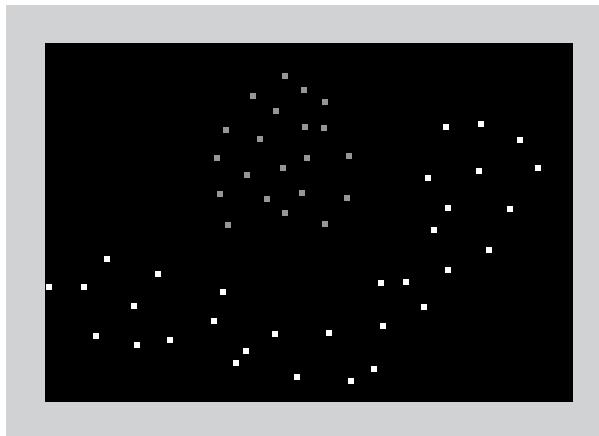
Fixeu-vos ara en la figura 43. En la segona imatge tenim un punt groc a la zona dels blaus. Gràcies a haver creat un marge flexible, l'hem deixat mal classificat, però tenim un marge igual que l'anterior. Els avantatges que obtenim tenen a veure amb l'error de generalització. Segurament aquest cas el trobaríem com un error en el conjunt d'entrenament.

Figura 43. Marge flexible amb *svmtoy*



En aquest punt veurem exemples de l'aplicació de *kernels*. Per començar, fixeu-vos en el conjunt de punts de la figura 44. Quin hiperplà pot separar aquests conjunts de punts? Un hiperplà lineal no els separaria. L'aplicació de marges flexibles deixaria massa punts mal classificats, i encara que fossin pocs, una línia recta no és un model correcte per a aquest problema. La solució per a aquest cas implica utilitzar un *kernel* per a fer una projecció a un altre espai.

Figura 44. Punts per a *kernels* amb *svmtoy*



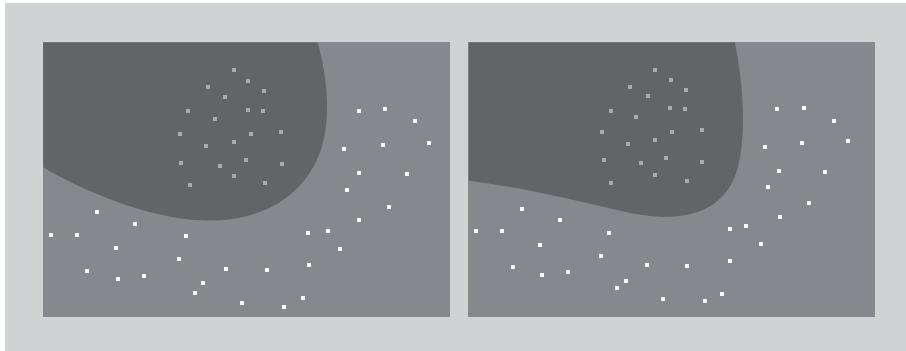
En la figura 45 teniu dues aproximacions amb diferents *kernels* polinòmics per a separar aquests conjunts de punts i, en la figura 46 dos, amb *kernels* radials. Els seus paràmetres exactes són, respectivament:

- polinòmic de grau dos ($-t 1 -d 2 -c 10000$)
- polinòmic de grau quatre ($-t 1 -d 4 -c 10000$)

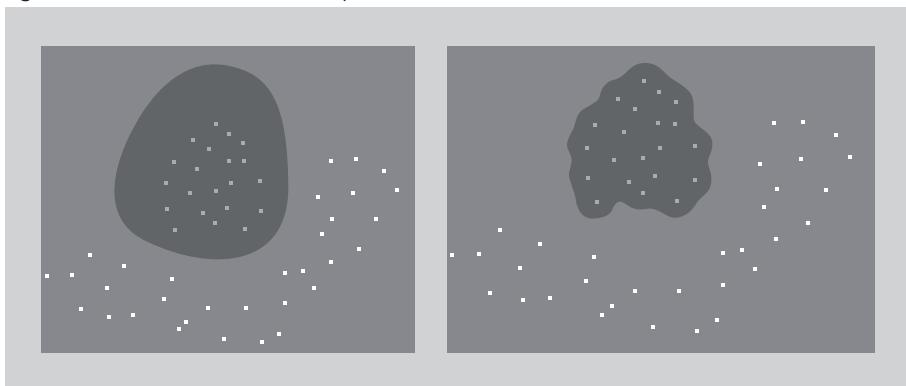
Kernel polinòmic

Un *kernel* polinòmic és de l'estil: $(\gamma \times u' \times v + coef0)^{\text{grau}}$ en què u' és el nostre vector d'atributs, v és un altre vector i la resta són valors reals. Se li poden passar com a paràmetres: $-t 1 -d \text{ grau} -g \gamma -r coef0$.

- radial amb gamma 50 ($-t 2 -g 50 -c 10000$)
- radial amb gamma 1000 ($-t 2 -g 1000 -c 10000$)

Figura 45. Kernels polinòmics amb *svmtoy***Kernel radial**

Un *kernel* radial de l'estil: $e^{-\gamma \times |u' \times v|^2}$ en què u' és el nostre vector d'atributs, v és un altre vector i la resta són valors reals. Se li poden passar com a paràmetres: $-t 2 -g \gamma$.

Figura 46. Kernels radials amb *svmtoy*

La figura 47 mostra un tercer exemple per treballar. En aquest cas els *kernels* polinòmics no ens serviran. Separarem els conjunts de punts utilitzant *kernels* radials. Fixeu-vos en l'efecte del paràmetre gamma que mostra la figura 48:

- radial amb gamma 1000 ($-t 2 -g 1000 -c 10000$)
- radial amb gamma 10 ($-t 2 -g 10 -c 10000$)

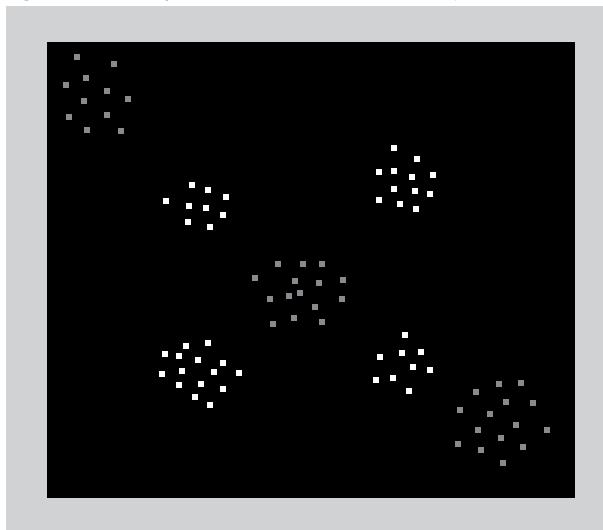
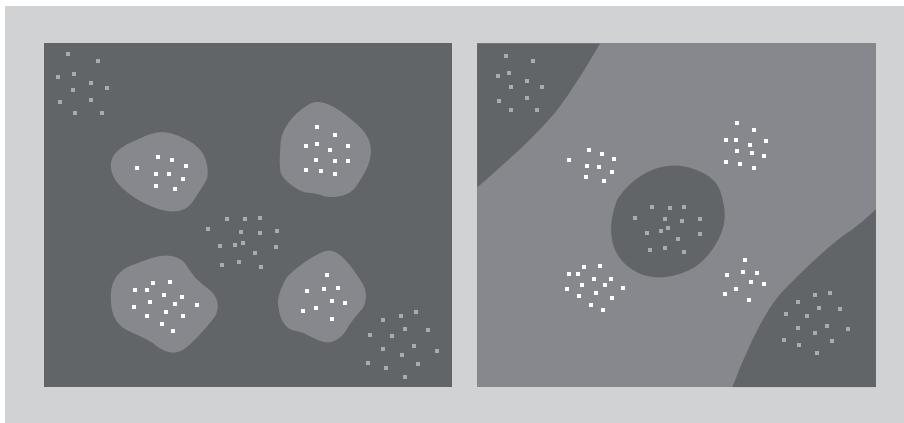
Figura 47. Punts per a *kernels* radials amb *svmtoy*

Figura 48. Kernels radials amb *svmtoy*

Formalització

Les màquines de vectors de suport es basen en el principi de la minimització del risc estructural* de la teoria de l'aprenentatge estadístic. En la seva forma bàsica, aprenen un hiperplà lineal de marge màxim que separa un conjunt d'exemples positius d'un d'exemples negatius. S'ha demostrat que aquest biaix d'aprenentatge té molt bones propietats tenint en compte la generalització dels classificadors induïts. El classificador lineal queda definit per dos elements: un vector de pesos w (amb un component per cada atribut) i un llindar b , que està relacionat amb la distància de l'hiperplà a l'origen: $h(x) = \langle x, w \rangle + b$ (figures 34 y 36). La regla de classificació $f(x)$ assigna +1 a un exemple x quan $h(x) \geq 0$ i -1 quan és menor. Els exemples positius i negatius més propers a l'hiperplà (w, b) són els anomenats *vectors de suport*.

* En anglès, *structural risk minimisation*. Descrita a: V. N. Vapnik (1998). *Statistical Learning Theory*. John Wiley.

Aprendre l'hiperplà de marge màxim (w, b) es pot abordar com un problema d'*optimització quadràtica convexa** amb una única solució. Això consisteix, en la seva *formulació primal* en:

* En anglès, *convex quadratic optimisation*. Aquest és un algorisme d'optimització matemàtica com els que es descriuen en el subapartat 5.2.

$$\text{minimitzar : } \|w\|$$

$$\text{subjecte a : } y_i(\langle w, x_i \rangle + b) \geq 1, \forall 1 \leq i \leq N$$

en què N és el nombre d'exemples d'entrenament i les y_i corresponen a les etiquetes.

L'anterior se sol transformar a la seva *formulació dual*, en què la solució es pot expressar com una combinació lineal d'exemples d'entrenament (vistos com a vectors):

$$\text{maximitzar : } \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

$$\text{subjecte a : } \sum_{i=1}^N y_i \alpha_i = 0, \alpha_i \geq 0, \forall 1 \leq i \leq N$$

en què els α_i són valors escalars no negatius determinats en resoldre el problema d'optimització.

D'aquesta formulació, el vector ortogonal a l'hiperplà queda definit com:

$$h(x) = \langle w, x \rangle + b = b + \sum_{i=1}^N y_i \alpha_i \langle x_i, x \rangle$$

en què $\alpha_i \neq 0$ per a tots els exemples d'entrenament que estan en el marge (vectors de suport) i $\alpha_i = 0$ per als altres. $f(x) = +1$ quan $h(x) \geq 0$ i -1 quan és menor defineix el classificador en la seva forma dual.

Encara que les màquines de vectors de suport són lineals en la seva forma bàsica, poden ser transformades en una *forma dual*, en la qual els exemples d'entrenament només apareixen dins de productes escalars, i permet l'ús de *funcions nucli** per a produir classificadors no lineals. Aquestes funcions treballen de manera molt eficient en espais d'atributs de dimensionalitat molt elevada, en què els nous atributs es poden expressar com a combinacions de molts atributs bàsics**.

* En anglès, *kernel functions*.

** Vegeu N. Cristianini; J. Shawe-Taylor (2000). *An Introduction to Support Vector Machines*. Cambridge University Press.

Si suposem que tenim una transformació no lineal ϕ de \mathbb{R}^D a algun espai de Hilbert \mathcal{S} , podem definir un producte escalar amb una funció nucli de tipus $\kappa(x, y) = \langle \phi(x), \phi(y) \rangle$ i obtenir una formulació dual:

$$\text{maximitzar : } \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N y_i y_j \alpha_i \alpha_j \kappa(x_i, x_j)$$

$$\text{subjecte a : } \sum_{i=1}^N y_i \alpha_i = 0, \alpha_i \geq 0, \forall 1 \leq i \leq N$$

i obtenir el vector ortogonal a l'hiperplà:

$$h(x) = \langle w, \phi(x) \rangle + b = b + \sum_{i=1}^N y_i \alpha_i \kappa(x_i, x)$$

Per a alguns conjunts d'entrenament no és desitjable obtenir un hiperplà perfecte, i és preferible permetre alguns errors en el conjunt d'entrenament per a obtenir hiperplans "millors" (figures 40 i 43). Això s'aconsegueix amb una variant del problema d'optimització, anomenada *marge flexible*, en què la con-

tribució a la funció objectiu de la maximització del marge i els errors en el conjunt d'entrenament es poden equilibrar per mitjà d'un paràmetre normalment anomenat C . Aquest paràmetre afecta les variables ξ_i en la funció (figura 40). Aquest problema queda formulat com:

$$\text{minimitzar : } \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^N \xi_i$$

$$\text{subjecte a : } y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall 1 \leq i \leq N$$

Anàlisi del mètode

Les màquines de vectors de suport van ser donades a conèixer l'any 92 per Boser, Guyon i Vapnik, però no van començar a guanyar popularitat en l'àrea de l'aprenentatge automàtic fins al final dels noranta.

Actualment, són un dels algorismes d'aprenentatge supervisats més utilitzats en la investigació. Han demostrat ser molt útils i s'han aplicat a una gran varietat de problemes relacionats amb el reconeixement de patrons en bioinformàtica, reconeixement d'imatges, categorització de textos, entre altres. S'ha provat que el seu biaix d'aprenentatge és molt robust i que s'obtenen molt bons resultats en molts problemes. S'estan dedicant també molts esforços en la comunitat investigadora internacional en les possibilitats que ha obert l'ús de *kernels*.

Quan apliquem aquest algorisme, hi ha una qüestió que hem de tenir en compte: estem combinant l'ús de *kernels* amb un algorisme d'optimització. Si apliquem un *kernel* que no encaixa bé amb les dades, pot passar que l'algorisme trigui a convergir o que doni una mala solució. El temps que triga a construir el model i el nombre de vectors que genera ens donen pistes de la bondat de la solució en la pràctica i de la separabilitat del conjunt de dades amb el *kernel* donat. El nombre de vectors de suport és un nombre inferior al del conjunt d'entrenament. La diferència entre els dos conjunts sol ser directament proporcional a la bondat de la solució i de l'hiperplà. El temps d'execució del procés d'aprenentatge sol ser inversament proporcional a la bondat.

Tant les SVM com els mètodes descrits en aquest subapartat són binaris. És a dir, només permeten separar entre dues classes. Quan ens trobem davant d'un problema que té més de dues classes, hem de crear un marc per a gestionar les prediccions de totes les classes a partir dels classificadors binaris. Hi ha diversos mecanismes de binarització per a abordar aquesta qüestió; aquí descriurem la tècnica de l'*un contra tots**¹, que és una de les més utilitzades en la bibliografia.

Referència bibliogràfica

B. Boser; I. Guyon; V. Vapnik (1992). A training Algorithm for Optimal Margin Classifiers. En les actes del *Workshop on Computational Learning Theory*. COLT

* En anglès, *one vs all*.

Aquesta tècnica consisteix a entrenar un classificador per classe, agafant com a exemples positius els exemples de la classe i com a negatius els exemples de totes les altres classes. El resultat de la classificació és un pes (nombre real per classe). Per al cas monoetiqueta, s'agafa com a predicció la classe amb el pes més gran. Per al cas multietiqueta, s'agafen com a prediccions totes les classes amb pesos positius.

Lectura complementària

Per a aprofundir en el tema de la binarització podeu consultar:

E. Allwein; R. E. Schapire; Y. Singer (2000). "Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers". *Journal of Machine Learning Research* (núm. 1, pàg. 113-141).

S. Har-Peled; D. Roth; D. Zimak (2002). "Constraint Classification for Multiclass Classification and Ranking". *Actas del Workshop on Neural Information Processing Systems*.

Ús en Python

Les dues implementacions d'SVM de lliure distribució més utilitzades són l'*svm^{light}* de Thorsten Joachims* i la *libsvm* de Chih-Chung Chang i Chih-Jen Lin**.

SVM^{light} és una implementació lliure de les SVM disponible per a la majoria de plataformes, que suporta problemes de regressió i classificació binària, la creació de marges flexibles, aprenentatge inductiu i transductiu, diferents tipus de *kernels* predefinitos, i permet que l'usuari defineixi els seus propis. La web disposa, a més, d'una extensió *SVM^{struct}* preparada per a crear ampliacions de *kernels* complexos i altres aplicacions. Com a exemple de l'extensió es pot trobar el tractament multiclasse i enllaços amb Python.

libsvm és una implementació de distribució lliure que suporta menys funcions, però que està disponible per a més sistemes (com *dll*, *jar*...). Una de les funcions que sí que té i que no té l'anterior és la possibilitat de treballar directament amb la matriu *kernel*. Conté l'*svmtoy* que hem utilitzat anteriorment per a donar la interpretació geomètrica de les SVM. En la web disposa d'algunes utilitats, enllaços amb altres llenguatges com Python i un *script* per a fer el tractament multiclasse.

Totes dues distribucions permeten treballar mitjançant llibreria i amb crides al sistema operatiu. A tall d'exemple, el codi 4.11 mostra l'ús de l'*svm^{light}* utilitzant crides al sistema operatiu i el codi 4.12 mostra l'ús de la *libsvm* per mitjà de la llibreria. El primer s'ha executat en un Mac OS X. En altres plataformes se'n pot veure alterat el codi.

* <http://svmlight.joachims.org>
 T. Joachims (1999). *Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning*. A: B. Schölkopf; C. Burges; A. Smola (ed.). MIT-Press.

** <http://www.csie.ntu.edu.tw/~cjlin/libsvm>
 Chih-Chung Chang; Chih-Jen Lin (2001). *LIBSVM: a library for support vector machines*.

Aprendentatge transductiu

L'aprendentatge transductiu utilitza exemples amb i sense etiquetes en el procés d'aprendentatge.

Codi 4.11: *svmlight* via crides al sistema operatiu

```

1 import os
2
3 parametres = '-t_1_-d_1'
4 train = 'train.dat'
5 test = 'test.dat'
6 model = 'model.svmlight.txt'
7 preds = 'prediccions.txt'
8 outs = 'output.txt'
9
10 os.system('./svm_learn' + parametros + ' ' + train + ' ' + model)
11
12 os.system('./svm_classify' + ' ' + test + ' ' + model + ' ' + preds +
13           ' >' + outs)
14
15 l = list(map(lambda l: l.strip(),
16               open(outs, 'r').readlines()))
17 print(l[0])
18 print(l[2])
19 print(l[3])

```

Codi 4.12: *libsvm* mitjançant llibreria

```

1 from libsvm.svmutil import *
2 from time import clock
3
4 train = list(map(lambda l: (l.strip()).split(' '),
5                  filter(lambda x: x[0] != '#',
6                        open('train.dat', 'r').readlines())))
7 y = list(map(lambda a: int(a.pop(0)), train))
8 x = [dict([(int(k), float(v))
9            for k, v in (l.split(':') for l in t)])
10      for t in train]
11 del(train)
12
13 inici = clock()
14 model = svm_train(y, x, '-c_4_-t_0')
15 print("learn", clock() - inici, "s")
16 del(y, x)
17 svm_save_model('model.libsvm.txt', model)
18
19 test = list(map(lambda l: (l.strip()).split(' '),
20                  filter(lambda x: x[0] != '#',
21                        open('test.dat', 'r').readlines())))
22 yt = list(map(lambda a: int(a.pop(0)), test))
23 xt = [dict([(int(k), float(v))
24            for k, v in (l.split(':') for l in t)])
25      for t in test]
26 del(test)
27
28 p_label, p_acc, p_val = svm_predict(yt, xt, model)

```

Si apliquem les *svm^{light}* per mitjà del codi 4.11 al corpus de la detecció de documents d'*adquisicions corporatives* obtenim una precisió del 97,67%. Triga 0,74 segons de CPU a calcular el model.

Si apliquem les *libsvm* per mitjà del codi 4.12 al mateix conjunt de dades obtenim una precisió del 97,33%. Triga 1,7 segons de CPU a calcular el model.

Ús d'un *kernel* definit per l'usuari amb les SVM

Si volem crear un *kernel* nou com els que vam veure en el subapartat anterior amb les *svm^{light}*, l'hem d'implementar en C en els arxius *kernel.h* i *kernel.c* que accompanyen la distribució i recompilar-lo.

Amb la *libsvm*, podem utilitzar l'opció de *kernel precalculat**. En la documentació ens diu que el format ha de ser tal com mostra la taula 32, en què L correspon al nombre d'exemples d'entrenament, $\langle classe \rangle$ serà $+1$ o -1 i en lloc del $?$ podem posar qualsevol cosa. Hem d'escollar com a *kernel* el número 4. El llistat 4.13 mostra el resultat d'aplicar tot això al corpus anterior.

* En anglès, *precomputed kernel*.

Taula 32. Exemple de representació d'un document en *libsvm* per al *kernel* precalculat

exemple d'entrenament x_i
$\langle classe \rangle \ 0 : i \ 1 : \kappa(x_i, x_1) \dots L : \kappa(x_i, x_L)$
exemple de test x
$\langle classe \rangle \ 0 : ? \ 1 : \kappa(x, x_1) \dots L : \kappa(x, x_L)$

Codi 4.13: SVM en *libsvm* amb matriu de *kernel*

```

1 from libsvm.svmutil import *
2 from time import clock
3
4 def kernelVSM(a, b):
5     la = [(int(k),float(v)) for k, v in (l.split(':') for l in a)]
6     lb = [(int(k),float(v)) for k, v in (l.split(':') for l in b)]
7     s = 0
8     i = 0
9     j = 0
10    while (i < len(la)) and (j < len(lb)):
11        if la[i][0] == lb[j][0]:
12            s += la[i][1] * lb[j][1]
13            i += 1
14            j += 1
15        elif la[i][0] < lb[j][0]:
16            i += 1
17        else:
18            j += 1
19    return s
20
21 train = list(map(lambda l: (l.strip()).split(' '),
22                  filter(lambda x: x[0] != '#',
23                         open('train.dat', 'r').readlines())))
24 y=list(map(lambda x: int(x.pop(0)), train))
25
26 inici = clock()
27 x = [dict([(i+1,kernelVSM(train[i], train[j])) for i in range(len(train))]) for j in range(len(train))]
28 list(map(lambda l, i: l.update({0:i+1}), x, range(len(x))))
29 print("km", clock() - inici, "s", len(x), "x", len(x[0]))
30
31 inici = clock()
32 model = svm_train(y, x, '-s 0 -t 4')
33 print("learn", clock() - inici, "s")
34 svm_save_model('model.km.txt', model)
35
36 ##model = svm_load_model('model.km.txt')
37
38 test = list(map(lambda l: (l.strip()).split(' '),
39                  filter(lambda x: x[0] != '#',
40                         open('test.dat', 'r').readlines())))
41 yt=list(map(lambda x: int(x.pop(0)), test))
42 inici = clock()
43 xt = [dict([(i+1,kernelVSM(test[j], train[i])) for i in range(len(train))]) for j in range(len(test))]
44
45
46
47

```

```

48 list(map(lambda i: i.update({0:-1}), xt, range(len(xt))))
49 print("kmt", clock() - inici, "s", len(xt), "x", len(xt[0]))
50
51 inici = clock()
52 p_label, p_acc, p_val = svm_predict(yt, xt, model)
53 print("predict", clock() - inici, "s")

```

En executar aquest programa, obtenim una precisió del 97,7%, amb uns temps: 31 minuts, 7 segons, 9 minuts i 2 segons, respectivament, per a: la construcció de la matriu de *kernel*, el procés d'aprenentatge, la construcció de la matriu de test i el procés de classificació.

4.6. Protocols de test

Aquest subapartat està dedicat a la validació dels processos d'aprenentatge, les mesures d'avaluació de sistemes de classificació i la significació estadística.

4.6.1. Protocols de validació

La validació és el procés pel qual comprovem com de bé (o malament) ha après un conjunt de dades un algorisme de classificació.

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 sobre el 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 anglès, *training*.

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 sobre el conjunt de test. Un error de mètode bastant usual és utilitzar el conjunt de test per a fer l'ajust de paràmetres; això no és estadísticament correcte. Una de les tècniques més utilitzades per a ajustar paràmetres són els algorismes d'optimització com els algorismes genètics, que es descriuen en el subapartat 5.4.

* El conjunt de validació per a l'ajust de paràmetres no té res a veure amb el procés de validació dels algorismes d'aprenentatge.

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 hagim obtingut. És a dir, diferents particions d'exemples condueixen a diferents resultats. En molts casos, els investigadors deixen disponibles els conjunts ja dividits en entrenament i test perquè les comparacions entre sistemes siguin més fiables.

Per a minimitzar aquest efecte, se sol utilitzar la *validació creuada** en lloc de la simple. Aquest tipus de validació consisteix a dividir un conjunt en k subconjunts. 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***. Aquest cas és com l'anterior definint la k com el nombre d'exemples del conjunt total. Ens quedaríam el conjunt de test amb un únic exemple. Aquest mètode no se sol utilitzar a causa del seu alt cost computacional.

* En anglès, *cross-validation* o *k-fold cross-validation*.

** En anglès, *leave one out*.

4.6.2. Mesures d'avaluació

Un dels problemes que ens trobem en descriure 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 subapartat 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 33 mostra el contingut de les matrius de confusions per al cas binari. A tall d'exemple, la cel·la *positiu verdader* 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 *false alarms* 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 33. Matriu de confusió

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

L'*error* o *ràtio d'error* mesura el nombre d'exemples que s'han classificat incorrectament sobre el total d'exemples. Es pretén que els algorismes d'aprenentatge tendeixin a minimitzar aquesta mesura. La fórmula és:

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

L'*exactitud** (de vegades anomenada també *precisió*) correspon als exemples que s'han classificat correctament sobre el total d'exemples. Aquesta mesura és la complementària de l'anterior. La fórmula correspon a:

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

La *precisió** o *valor predictiu positiu* correspon als exemples positius ben classificats sobre el total d'exemples amb predicció positiva. La fórmula és

* En anglès, *accuracy*.

* En anglès, *precision*.

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

La *sensibilitat** correspon als exemples positius ben classificats sobre el total d'exemples positius. La fórmula és:

* En anglès, *recall*.

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

El conjunt de precisió i sensibilitat es pot veure com una versió estesa de l'*exactitud*. Aquestes mesures vénen 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 ens arriba també la mesura *F1*, proposada el 1979 per van Rijsbergen, que combina les dues anteriors:

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

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*:

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

que se solen utilitzar en l'àrea de la teoria de detecció de senyals. D'aquí surt també un diagrama anomenat *ROC* que s'utilitza en diversos àmbits com el de la biotecnologia.

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

Lectura complementària

T. Fawcet (2006). An Introduction to ROC Analysis. *Pattern Recognition Letters* (núm. 27, pàg. 861-874).

4.6.3. Tests estadístics

Quan tenim les prediccions de dos classificadors i hem aplicat alguna de les mesures d'avaluació, estadísticament no és suficient mirar quin ha obtingut un valor més alt; cal mirar, a més, si la diferència és estadísticament significativa.

Per a fer això, hi ha dues maneres en funció del tipus de validació que hagim utilitzat. Si hem utilitzat la validació creuada, hem d'usar el *test de Student* i si hem utilitzat la validació simple el *test de McNemar*. Donats dos classificadors F i G :

- El test de *Student* amb una confiança de $t_{9, 0,975} = 2,262$ o equivalent (depèn del nombre de particions) en la validació creuada:

$$\frac{\bar{p} \times \sqrt{n}}{\sqrt{\frac{\sum_{i=1}^n (p(i) - \bar{p})^2}{n-1}}} > 2,262$$

en què n és el nombre de particions, $\bar{p} = \frac{1}{n} \sum_{i=1}^n p(i)$, $p(i) = p_F(i) - p_G(i)$, i $p_X(i)$ és el nombre d'exemples mal classificats pel classificador X (en què $X \in \{F, G\}$).

- El test de *McNemar* amb un valor de confiança de $\chi^2_{1, 0,95} = 3,842$ amb la validació simple:

$$\frac{(|A - B| - 1)^2}{A + B} > 3,842$$

en què A és el nombre d'exemples mal classificats pel classificador F però no pel G , i B és el nombre d'exemples mal classificats pel G però no pel F .

Demšar, en l'article “Statistical Comparisons of Classifiers over Multiple Data Sets”, recomana com a alternativa el test no paramètric de Wilcoxon:

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}}$$

Lectura complementària

J. Demšar (2006). “Statistical Comparisons of Classifiers over Multiple Data Sets”. *Journal of Machine Learning Research* (núm. 7, pàg. 1-30).

en què N és el nombre de conjunts de dades i $T = \min(R^+, R^-)$, en què:

$$R^+ = \sum_{d_i > 0} rank(d_i) + \frac{1}{2} \sum_{d_i=0} rank(d_i)$$

$$R^- = \sum_{d_i < 0} rank(d_i) + \frac{1}{2} \sum_{d_i=0} rank(d_i)$$

tenint en compte que d_i és la diferència de la mesura d'avaluació entre dos classificadors sobre el conjunt de dades i i que fem un rànquing sobre aquestes distàncies. Els rànquings tal que $d_i = 0$ es divideixen equitativament entre les dues sumes. En cas de ser un nombre ímpar, se n'ignora una. Per a $\alpha = 0,05$, la hipòtesi nul·la es rebutja quan z és més petit que $-1,96$.

4.6.4. Comparativa de classificadors

En aquest subapartat anem a veure dues maneres de comparar classificadors entre ells.

Comparatives de més de dos classificadors sobre múltiples conjunts de dades

La manera clàssica de fer una comparativa de més de dos classificadors sobre múltiples conjunts de dades és utilitzar el mètode estadístic ANOVA. Desafortunadament, aquest estadístic fa moltes suposicions que usualment no es compleixen en aprenentatge automàtic. Demšar ens recomana un test alternatiu, el de Friedman.

Suposem que tenim k classificadors i n conjunts de dades. Per a calcular el test de Friedman fem un rànquing dels diferents algorismes per a cada conjunt de dades per separat. En cas d'empat amitjanem els rànquings. r_i^j correspondrà al rànquing de l'algorisme j sobre el conjunt de dades i . Sota la hipòtesi nul·la, l'estadístic es calcula com:

$$\chi_F^2 = \frac{12n}{k(k+1)} \left(\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right)$$

en què $R_j = \frac{1}{n} \sum_i r_i^j$ correspon a la mitjana dels algorismes sobre el conjunt de dades i .

Lectures complementàries

J. Demšar (2006). “Statistical Comparisons of Classifiers over Multiple Data Sets”. *Journal of Machine Learning Research* (núm. 7, pàg. 1-30).

Aquest estadístic es distribueix d'acord amb χ^2_F amb $k - 1$ graus de llibertat, quan n i k són suficientment grans ($n > 10$ i $k > 5$)*.

L'estadístic següent** és una millora de l'anterior (no tan conservador):

$$F_F = \frac{(n-1)\chi^2_F}{n(k-1) - \chi^2_F}$$

que es distribueix de sobre una distribució F amb $k - 1$ i $(k - 1)(n - 1)$ graus de llibertat.

* Per a valors menors, consulteu els valors crítics exactes a:
J. H. Zar (1998). *Biostatistical Analysis* (4a. ed.). Prentice Hall.
D. J. Sheskin (2000). *Handbook of parametric and nonparametric statistical procedures*. Chapman & Hall/CRC.

** R. L. Iman; J. M. Davenport (1980). "Approximations of the critical region of the Friedman statistic". *Communications in Statistics*.

Mesures d'acord i kappa

En aquest subapartat anem a veure dues de les mesures més útils per a comparar el comportament de diferents classificadors.

Donades les prediccions de dos classificadors F i G , l'*acord** (observat) es calcula com:

$$Pa = \frac{A}{N}$$

* En anglès, *agreement*.

en què A correspon al nombre d'exemples per als quals els dos han donat la mateixa predicció i N és el nombre total d'exemples.

I l'estadístic *kappa* com:

$$\kappa(F, G) = \frac{Pa - Pe}{1 - Pe}$$

en què Pe (l'*agreement* esperat) es calcula com:

$$Pe = \sum_{i=1}^{ns} \frac{P_F(i) \times P_G(i)}{N^2}$$

en què ns és el nombre de classes, i $P_X(i)$ és el nombre de prediccions del classificador X (amb $X \in \{F, G\}$) de la classe i .

Vegeu també

Aquest estadístic té en compte com es distribueixen les prediccions entre les diferents classes els classificadors. A partir d'aquest, es pot estudiar com es comporten diferents classificadors entre ells i fins i tot crear-ne un dendrograma. En un experiment fet sobre un problema de llenguatge natural*, dóna com resultat una semblança notable entre els algorismes de l'AdaBoost i les

En el subapartat 2.3.3. s'estudia la creació de dendrogrames.

* Capítol 7 del llibre: E. Agirre; P. Edmonds (ed.) (2006) *Word Sense Disambiguation: Algorithms and Applications*. Springer.

SVM diferenciant-los de la resta, per posar solament un exemple. L'experiment parteix d'una taula com la 34, en què c_i correspon al classificador i .

Taula 34. Matriu de kappa de dues en dues

	c_1	c_2		c_{N-1}
c_2	$\kappa(c_1, c_2)$	–	–	–
c_3	$\kappa(c_1, c_3)$	$\kappa(c_2, c_3)$	–	–
:	:	:	..	:
c_N	$\kappa(c_1, c_N)$	$\kappa(c_2, c_N)$	–	$\kappa(c_{N-1}, c_N)$

Conclusions

Hi ha dos conceptes que afecten profundament tots els processos d'aprenentatge. El primer és l'anomenada *maledicció de la dimensionalitat**. Aquest concepte té a veure amb la necessitat d'exemples d'entrenament necessaris per a aprendre en funció de la complexitat de les regles de classificació; que augmenta de manera exponencial.

* En anglès, *curse of dimensionality*.

El segon concepte és el biaix inductiu*. Aquest concepte preveu que els diferents algorismes d'aprenentatge afavoreixen diferents tipus de regles de classificació en els processos d'aprenentatge.

* En anglès, *inductive bias*.

Seria desitjable disposar d'un algorisme d'aprenentatge que fos millor que tots els altres, independentment del conjunt de dades per tractar. Desafortunadament, això no ocorre i fa que necessitem diferents algorismes per a diferents tipus de conjunts de dades. Aquests resultats s'han quantificat i s'han denominat, en anglès, *the no free lunch theorem* (Kulkarni i Harman, 2011).

Tot això ha fet que molts autors considerin que el disseny d'aplicacions pràctiques de processos d'aprenentatge pertany tant a l'enginyeria i a la ciència com a l'art.

5. Optimització

5.1. Introducció

L'**optimització** d'un problema P és la tasca de buscar els valors que apliquats a P satisfacin unes determinades expectatives. Dependent de la naturalesa de P les expectatives que han de complir aquests valors, o sigui, el que cal buscar en la tasca d'optimització, pot tenir característiques molt diferents; en qualsevol cas, se suposa que són les característiques més desitjables per a qui proposa el problema.

Així, es poden aplicar tècniques d'optimització per a trobar valors que minimitzin els costos d'una activitat, que minimitzin el temps requerit per a dur-la a terme, que determinin la trajectòria òptima d'un vehicle espacial, que gestionin de manera òptima un conjunt de recursos com una xarxa de serveis, que maximitzin els beneficis d'una inversió o que trobin la combinació de gens que més s'associa a una determinada malaltia, entre innombrables exemples possibles.

Per a cada problema P es defineix una **funció objectiu** f que mesura d'alguna manera la idoneïtat de cada possible solució de P : cost, temps requerit, benefici, combustible emprat, temps de resposta d'un sistema, productivitat per hora, etc. El domini de f , o sigui el conjunt de punts que es poden provar com a solució al problema, rep el nom d'**espai de solucions** i es representa amb la lletra S .

El rang de f sol ser un valor escalar (enter o real, o un subconjunt), si bé en alguns casos es pot tractar d'un valor de més dimensions. Dependent de P i de la manera com es defineixi f la tasca d'optimització pot consistir a trobar els punts que minimitzin f (cost, temps requerit, etc.) o els punts que la maximitzin (benefici, productivitat, eficiència).

Sovint la funció f es defineix com una **funció d'error**, que és la diferència entre el resultat desitjat i l'obtingut per un punt $x \in S$. En aquests casos l'objectiu de la tasca d'optimització sempre és, lògicament, minimitzar f .

En qualsevol cas, cal diferenciar l'enunciat del problema P de la funció objectiu f , ja que sovint convé definir f de manera substancialment diferent a P per a incrementar l'efectivitat dels mètodes d'optimització.

Si veiem f com el relleu geogràfic d'un país, amb valls i muntanyes, és important que f sigui una funció sense grans planes, grans zones de valor homogeni que no donen cap orientació sobre el millor camí per prendre. L'aigua s'estanca en les planes, només flueix i busca el mar en els pendent. f s'ha de definir de manera que reflecteixi els més petits canvis en la qualitat d'una solució, perquè els mètodes d'optimització sàpiguen que, a poc a poc, estan millorant la solució proposada. En el subapartat 5.7. es planteja un exemple, l'acoloriment d'un mapa, en el qual s'aprecia clarament aquest requisit. Si f és el nombre de colors necessari per a acolorir un mapa sense que hi hagi dos països contigus del mateix color, tindrem una funció objectiu amb vastes planes. No obstant això, si definim f com el nombre de països contigus del mateix color, tindrem una mesura molt més fina del progrés de l'optimització, ja que cada vegada que es resolgui un d'aquests "errors", f disminuirà una mica.

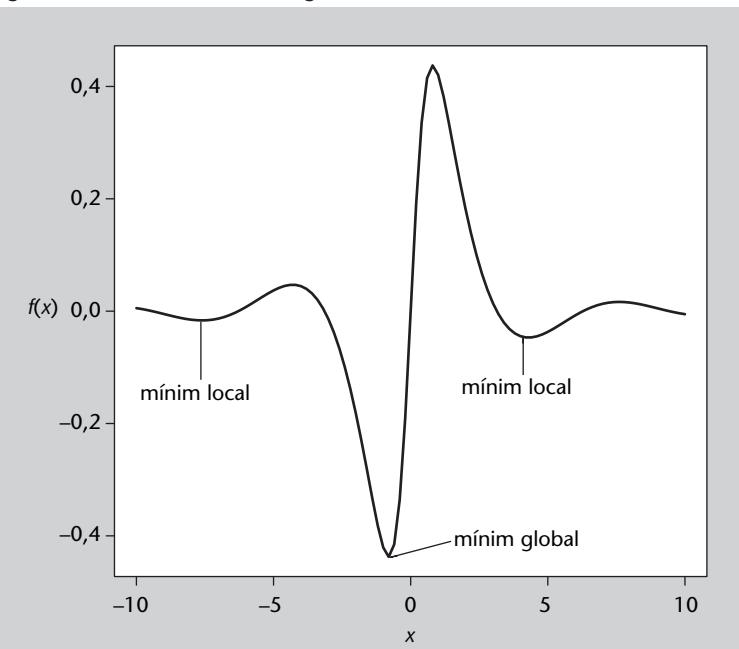
Aquest exemple ens porta a una última consideració sobre f : en molts problemes hi ha solucions prohibides, ja que incompleixen algunes restriccions del problema; en l'exemple anterior, un acoloriment en el qual dos països contigus tenen el mateix color no és vàlid. No obstant això, molt sovint convé definir una **f relaxada** respecte a P , que permeti tals solucions prohibides com a pas intermedi per a assolir les solucions òptimes desitjades. En cas contrari, el salt necessari per a passar d'una solució vàlida a una altra millor pot ser tan gran que resulti, en la pràctica, inassolible. Per exemple, permetent que dos països contigus tinguin el mateix color com a pas intermedi per a aconseguir un acoloriment vàlid. Si no es permet, el mètode ha d'obtenir un nou acoloriment del mapa en un sol pas, la qual cosa gairebé segur el condemnarà al fracàs.

Seguint amb l'analogia geogràfica de l'espai de solucions, i suposant que volem minimitzar f , el que es busca és la vall amb el punt més baix (en el cas contrari buscarem la muntanya més alta). Depenent del problema és possible que hi hagi una única vall amb la solució óptima o, per contra, que hi hagi nombroses valls amb solucions més o menys bones. Si ens trobem en el fons d'una vall ens sembla que la nostra posició és la més baixa, encara que és possible que hi hagi una altra vall més baixa encara en alguna altra part. En un problema d'optimització pot ocórrer el mateix, que hi hagi moltes zones amb solucions relativament bones, si bé en una altra zona de S es puguin trobar solucions encara millors. En aquest cas es parla d'**òptims locals i òptims globals**.

L'objectiu dels mètodes d'optimització és trobar l'òptim global i no "deixar-se enganyar" pels òptims locals, encara que això no sempre és fàcil o possible.

El principal obstacle és que, per a passar d'un òptim local a un òptim global (o a un òptim local millor que el primer), és necessari passar per punts pitjors (sortir d'una vall per a baixar a una altra), i en principi això es veu com un empitjorament de la solució, i per tant es descarta. Se'n pot veure un exemple en la figura 49.

Figura 49. Mínims locals i mínim global



Una última consideració es refereix a la mida de S . En gran part dels problemes reals, S té una mida molt gran, especialment si té moltes variables o dimensions. En el que es coneix com la **maledicció de la dimensionalitat***, si tenim un espai de solucions d'una dimensió (una recta) de costat 1, amb 100 punts es pot explorar en intervals de 0,01. No obstant això, si tenim un espai de dues dimensions i costat 1 (un quadrat), seran necessaris 100^2 punts per a poder explorar-lo amb una resolució de 0,01; en un cub es necessiten 100^3 punts, i en un hipercub de 10 dimensions seran necessaris 100^{10} punts. Per aquest motiu una exploració exhaustiva de S sol ser impracticable, i ni tan sols mètodes com el de Newton, que van subdividint l'espai de solucions progressivament, són pràctics.

Així, en molts casos no es coneix quina és la solució òptima; de fet, els mètodes d'optimització descrits en aquest apartat estan dissenyats per a trobar solucions acceptablement bones, ja que no és possible saber si són les òptimes, en espais de solucions molt grans (amb moltes dimensions).

* En anglès, *curse of dimensionality*, terme encunyat per Richard E. Bellman

Vegeu també

Una estratègia complementària consisteix a utilitzar les tècniques de reducció de dimensionalitat vistes en el subapartat 3.1. d'aquest mòdul.

5.1.1. Tipologia dels mètodes d'optimització

Hi ha una gran varietat de mètodes d'optimització a causa que cada problema requereix o permet estratègies diferents. Bàsicament es poden classificar en les categories següents:

- **Mètodes analítics:** consisteixen a fer alguna transformació analítica de la funció objectiu per a determinar els punts d'interès. Com a exemple s'estudia el mètode dels multiplicadors de Lagrange en el subapartat 5.2., encara que també hi ha altres mètodes, com la programació quadràtica.
- **Mètodes matemàtics iteratius:** apliquen una operació repetidament fins que es compleix un criteri determinat. Alguns exemples són el mètode de Newton i el mètode d'ascens (o descens) de gradients.
- **Mètodes metaheurístics:** es denominen **heurístiques** les tècniques basades en l'experiència o en analogies amb altres processos que resulten útils per a resoldre un problema. En l'àrea que ens ocupa, la de l'optimització, els heurístics són tècniques que solen ajudar a optimitzar un problema, habitualment prenen solucions aleatòries i millorant-les progressivament mitjançant l'heurístic en qüestió. Els mètodes que estudiarem en aquest text pertanyen a aquesta categoria (excepte el de Lagrange), ja que són els que es consideren propis de l'àrea de la intel·ligència artificial. Exemples: cerca aleatòria, algorismes genètics i programació evolutiva, cerca tabú, reacció simulada, etc.

El prefix *meta-* dels mètodes metaheurístics és perquè tals heurístics són aplicables a qualsevol problema, ja que els mètodes no pressuposen res sobre el problema que prenenen resoldre i, per tant, són aplicables a una gran varietat de problemes. En general es parlarà de *metaheurístic* o simplement de *mètode* per a parlar de cada tècnica en general, i d'*algorisme* per a l'aplicació d'una tècnica a un problema concret, si bé per motius històrics aquesta nomenclatura no sempre se segueix (com en el cas dels algorismes genètics).

Els mètodes metaheurístics no garanteixen l'obtenció de la solució òptima, però soLEN proporcionar solucions acceptablement bones per a problemes inabordables amb tècniques analítiques o més exhaustives.

5.1.2. Característiques dels metaheurístics d'optimització

La majoria de metaheurístics d'optimització parteixen d'una o més solucions triades aleatoriament que es van millorant progressivament. Si es parteix d'una única solució aleatòria i les noves solucions proposades per l'heurístic en depenen, es corre el risc de quedar-se estancat en un òptim local. Per aquest motiu tals mètodes se solen denominar **mètodes locals**, en contraposició dels

mètodes globals, que solen generar diverses solucions aleatòries independents i, per tant poden explorar diferents àrees de l'espai de solucions; així eviten, en la mesura del possible, limitar-se a un òptim local determinat.

Un altre aspecte comú als mètodes metaheurístics que s'estudiaran en aquest apartat és la seva naturalesa iterativa, la qual cosa lògicament porta a la qüestió del nombre d'iteracions que haurà d'executar cada algorisme. Quan l'òptim global sigui conegut (per exemple, en una funció d'error serà 0) i sigui possible assolir-lo en un temps acceptable, es podrà iterar l'algorisme fins a arribar a l'òptim global. No obstant això, en la majoria de problemes l'òptim global no és conegut o no es pot garantir que es trobarà en un temps determinat. Per tant, cal fixar algun criteri de terminació de l'algorisme. Entre els criteris més habituals es troben:

- Assolir un valor predeterminat de la funció objectiu (per exemple, un error $< \epsilon$).
- Un nombre fix d'iteracions.
- Un temps d'execució total determinat.
- Observar una tendència en les solucions obtingudes (no s'observa millora durant n iteracions).

Finalment, els mètodes metaheurístics se solen caracteritzar per una sèrie de paràmetres que s'han d'ajustar de manera adequada per a resoldre cada problema eficientment. En tractar-se d'heurístics no hi ha regles objectives que determinin tals paràmetres, i per tant es recorre a valors orientatius obtinguts en estudis experimentals sobre diferents tipus de problemes. De fet, en alguns casos s'aplica un algorisme d'optimització sobre els paràmetres de l'algorisme d'optimització, la qual cosa es coneix com a **metaoptimització**.

5.2. Optimització mitjançant multiplicadors de Lagrange

Un fabricant d'envasos ens planteja la qüestió següent: quines proporcions ha de tenir una llauna cilíndrica perquè contingui un volum determinat utilitzant la menor quantitat possible de metall? Què és millor? Que sigui molt alta, molt ampla o igual d'alta que d'amplia?

L'optimització mitjançant multiplicadors de Lagrange és un mètode d'optimització matemàtica que permet trobar el mínim o el màxim d'una funció tenint en compte una o més **restriccions**, que són una altra funció o funcions que s'han de satisfer.

En l'exemple anterior, la funció que volem minimitzar és la que determina la superfície de la llauna, i la restricció és la que en determina el volum. Noteu que, si no s'afegeix la restricció del volum, la minimització de la funció de la superfície donarà com a resultat una llauna d'altura i radi zero; un resultat correcte però poc útil en la pràctica.

5.2.1. Descripció del mètode

A continuació es descriurà el mètode aplicant-lo a funcions de dues variables; la generalització a més dimensions és immediata. Sigui $f(x,y)$ la funció que volem minimitzar, i sigui $g(x,y) = c$ la restricció que s'aplica. Sense entrar en l'explicació matemàtica, els punts que busquem són aquells en els quals els **gradients** de f i g coincideixen:

$$\nabla_{x,y}f(x,y) = \lambda \nabla_{x,y}g(x,y) \quad (48)$$

en què λ és una constant que és necessària per a igualar les longituds dels vectors gradient, ja que poden tenir la mateixa direcció però longituds diferents. La funció gradient d'una funció es calcula mitjançant les derivades parcials de la funció per a cadascuna de les seves variables:

Gradient d'una funció

El gradient d'una funció f en un punt és un vector que apunta en el màxim pendent de f . La funció gradient de f és un camp de vectors que apunten, en cada punt, en la direcció de màxim pendent de f , i es representa per ∇f .

$$\nabla_{x,y}f(x,y) = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right\rangle \quad (49)$$

$$\nabla_{x,y}g(x,y) = \left\langle \frac{\partial g}{\partial x}, \frac{\partial g}{\partial y} \right\rangle \quad (50)$$

Per tant, el problema es transforma en un sistema d'equacions que cal resoldre per a obtenir la solució desitjada:

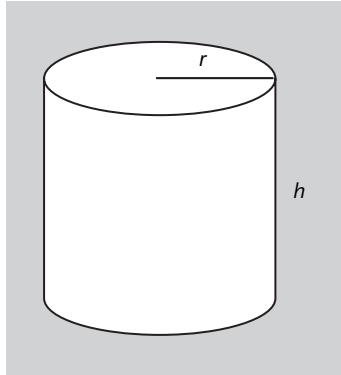
$$\left. \begin{array}{l} \frac{\partial f}{\partial x} = \lambda \frac{\partial g}{\partial x} \\ \frac{\partial f}{\partial y} = \lambda \frac{\partial g}{\partial y} \end{array} \right\} \quad (51)$$

En aquest sistema hi ha dues equacions i tres variables: x , y i λ . En general obtindrem els valors per a x i y en funció de λ , que és el paràmetre que ens dóna el grau de llibertat necessari per a triar els valors desitjats que compleixin la restricció imposta; en altres paraules, el nostre problema d'optimització s'ha transformat en una funció d'una única variable, λ , que ens permet triar el valor més adequat a les especificacions del problema.

5.2.2. Exemple d'aplicació

Tornem al problema de la llauna cilíndrica la superfície de la qual volem minimitzar per a un determinat volum. En la figura 50 es mostra la llauna amb les dues variables que la caracteritzen, el radi r i l'altura h .

Figura 50. Una llauna cilíndrica



En aquest cas, la funció que volem minimitzar és la superfície de la llauna (superfície del costat més superficial de les dues tapes), que és la que implica la quantitat de metall utilitzat, i la restricció és el volum contingut per la llauna, i llavors en aquest problema f i g seran:

$$f(x,y) = 2\pi rh + 2\pi r^2 \quad (52)$$

$$g(x,y) = \pi r^2 h \quad (53)$$

Després d'obtenir les funcions gradient i multiplicar ∇g per λ , cal resoldre el sistema d'equacions següent:

$$\left. \begin{array}{l} 2\pi rh = \lambda(2\pi h + 4\pi r) \\ \pi r^2 = \lambda 2\pi r \end{array} \right\} \quad (54)$$

Expressant r i h en funció de λ s'obté:

$$\left. \begin{array}{l} r = 2\lambda \\ h = 4\lambda \end{array} \right\} \quad (55)$$

I d'això es dedueix que la relació entre radi i altura d'una llauna ha de ser $h = 2r$ per a utilitzar la menor quantitat de material en relació amb el volum que conté.

5.2.3. Anàlisi del mètode

La naturalesa analítica d'aquest mètode en condiciona els avantatges i inconvenients. Com a avantatge tenim la possibilitat d'obtenir no una sinó un rang de solucions òptimes de manera relativament ràpida. D'altra banda, la seva principal limitació és que només és aplicable a funcions diferenciables, la qual cosa restringeix el tipus de problemes en els quals es pot utilitzar aquest mètode.

5.3. Recocció simulada

Suposem que tenim un sistema informàtic amb N servidors i C clients que necessiten accedir als servidors. Cada servidor té un temps d'espera base t_{eb} , si bé el temps d'espera efectiu t_{ee} de cada servidor depèn tant de t_{eb} com del nombre de clients en aquest servidor, C , segons l'expressió $t_{ee} = t_{eb}(1 + C)$. Així, si un servidor té $t_{eb} = 3$ i 5 clients, tindrem $t_{ee} = 3(1 + 5) = 18$.

En aquest problema es busca una distribució de C clients entre N servidors, cadascun amb un t_{eb} possiblement diferent, de manera que el temps d'espera efectiu mitjà per a tots els clients sigui el mínim.

La recocció simulada* és una metaheurística d'optimització amb l'objectiu de trobar una aproximació raonablement bona de l'òptim global d'una funció determinada. Hi ha, per tant, dues idees importants: primer que el que es pretén trobar és una aproximació del punt òptim global, donant per fet que buscar exhaustivament l'òptim global no és possible en un termini de temps raonable, atesa la mida de l'espai de cerca. En segon lloc, la recocció simulada està dissenyada per a evitar, en la mesura del possible, quedar embussada en òptims locals.

* En anglès, *simulated annealing*.

La recocció simulada s'inspira en la tècnica metal-lúrgica de la *recocció*, en la qual els metalls es reescalfen i tornen a deixar refredar diverses vegades perquè els seus àtoms es vagin recol·locant en estats de menys energia. Així es formen cristalls més grans, i per tant es redueixen les imperfeccions del metall i se'n milloren les propietats. Aquest procés es du a terme de manera decreixent: al principi el reescalfament aplicat és més gran per a eliminar els defectes de més mida, i progressivament, el reescalfament aplicat es redueix per a eliminar els defectes més fins.

De la mateixa manera, la recocció simulada parteix d'un estat a l'atzar dins de l'espai de cerca i va aplicant canvis en aquest estat per veure si la seva energia (en aquest cas, la funció que es vol minimitzar) augmenta o disminueix.

Amb la finalitat d'evitar quedar embussat en òptims locals, la recocció simulada permet canvis a estats de més energia (que en principi són pitjors) per a aconseguir sortir d'un òptim local a la cerca de l'òptim global. A mesura que avança l'execució, no obstant això, els empitjoraments que es permeten són cada vegada més petits, de manera que a poc a poc l'algorisme vagi trobant una aproximació raonablement bona a l'òptim global.

5.3.1. Descripció del mètode

L'objectiu bàsic de la recocció simulada és passar d'un estat qualsevol de l'espai de cerca, amb una energia probablement alta, a un estat de baixa energia. El terme *energia* se substituirà, en cada problema, per la mesura concreta que es vulgui minimitzar (error, valor d'una funció, temps de resposta, cost econòmic, etc.).

Es diu que un estat s' és veí d'un altre estat s si hi ha alguna transformació senzilla (mínima) per a passar de s a s' . En el cas d'espais de cerca discrets, aquesta transformació sol ser evident; per contra, si l'espai de cerca és continu, és necessari definir una distància màxima de transformació per a considerar que dos estats són veïns.

En el procés de recocció s'aplica un increment de temperatura més gran en les fases inicials, i a mesura que el procés avança aquest increment es va reduint. De la mateixa manera, en la recocció simulada es comença amb un marge de temperatura alt i progressivament es va reduint. D'aquesta manera, en les iteracions inicials és probable que l'algorisme accepti un nou estat s' encara que tingui una energia més alta que l'estat previ s ; no obstant això, a mesura que l'algorisme avança, és menys probable que s'accepti un estat amb energia més alta. La temperatura, per tant, decreix a mesura que l'algorisme avança; una formulació habitual és $T = ir * F$, en què ir és el nombre d'iteracions restants i F és un factor que permet ajustar la tolerància amb els nivells d'energia dels nous estats, qüestió que s'analitza més endavant.

El primer pas d'un algorisme que utilitza la recocció simulada consisteix a prendre un estat a l'atzar en l'espai de cerca. També és possible prendre un estat conegit que se sapiga que és millor que la mitjana. Denominarem s l'estat actual, i $E(s)$ serà el seu nivell d'energia.

En cada iteració de l'algorisme es genera aleatoriament un estat s' veí de l'estat actual s ; en funció del nivell d'energia de l'estat actual $e = E(s)$ i del nou estat

$e' = E(s')$, el sistema canvia a s' o per contra roman en s . La probabilitat d'acceptació del nou estat està determinada per l'expressió:

$$P(e, e', T) = \begin{cases} 1 & \text{si } e' < e \\ \exp((e - e')/T) & \text{si } e' \geq e \end{cases} \quad (56)$$

Com es veu, la temperatura T –i per tant, el nombre d'iteracions actual– influeix en la decisió d'acceptar o no un nou estat. Recordem que $T = ir * F$; F es pot utilitzar per a ajustar la probabilitat d'acceptació, atès que segons l'ordre de magnitud de l'energia d'un estat, un increment d'energia d'1 pot ser molt gran o molt petit. Una regla senzilla per a ajustar l'algorisme és $i * F \simeq \Delta E_{max}$, si i és el nombre total d'iteracions de l'algorisme i ΔE_{max} el màxim increment d'energia acceptable, ja que així la probabilitat inicial d'acceptació tindrà un valor proper al 50%.

En resum, en les primeres iteracions és probable que s'acceptin canvis d'estat que impliquin un increment de l'energia (allunyant-se de l'objectiu, que és minimitzar l'energia). D'aquesta manera s'afavoreix que el sistema surti d'un mínim local i intenti buscar un mínim global. No obstant això, a mesura que l'algorisme avança en l'execució es va fent menys probable que s'acceptin canvis d'estat amb increments d'energia, ja que se suposa que el que ha de fer l'algorisme és trobar el mínim de l'entorn en el qual es trobi.

Una millora habitual de l'algorisme consisteix a emmagatzemar el millor estat trobat en tota l'execució, i retornar aquest estat com a resultat final, encara que no es tracti de l'estat final assolut. Si bé aquesta millora no té correspondència amb el procés físic de recocció, és molt senzill implementar-la i pot millorar el comportament de l'algorisme en alguns casos.

5.3.2. Exemple d'aplicació

La recocció simulada s'utilitza freqüentment per a optimitzar la planificació i assignació de recursos de diversos tipus. En aquest cas l'aplicarem a un exemple senzill d'assignació de recursos, concretament al problema de l'assignació de clients a servidors enunciad anteriorment.

Per a desenvolupar aquest algorisme, considerarem que un estat s és una partició determinada dels C clients entre els N servidors; per exemple, si tenim 40 clients i 5 servidors, un possible estat és $s = (10, 4, 8, 7, 11)$. Un estat s' és veí de s si d'un a l'altre només s'ha mogut un client d'un servidor a l'altre. Un exemple d'estat veí de s seria $s' = (9, 5, 8, 7, 11)$, en el qual un client s'ha mogut del servidor 1 al 2.

En la taula 35 es mostra un exemple d'execució de la recocció simulada aplicada al problema d'assignació de clients a servidors. El nombre de servidors és $N = 5$, el de clients $C = 40$, els temps d'espera base dels servidors és $(5,7,7,2,1)$, i en l'estat inicial els clients es distribueixen per igual entre els cinc servidors, i llavors l'estat inicial és $s_i = (8,8,8,8,8)$. El nombre d'iteracions és 10, i el factor de càlcul de la temperatura és 0,1, de manera que els increments màxims d'energia (temps mitjà d'espera) siguin de l'ordre d'1,0. Noteu que quan un canvi d'estat no s'accepta, es roman en l'estat actual. Les iteracions estan numerades en ordre decreixent perquè coincideixin amb la formulació de l'algorisme.

Taula 35. Exemple d'assignació de clients a servidors

#Iter	A	B	C	D	E	T mitjà espera	$P(e,e',T)$	Acceptat
Inicial	8	8	8	8	8	39,6		
10	8	7	8	8	9	37,25	1	SÍ
9	8	7	9	8	8	39,95	0,05	NO
8	8	7	7	8	10	34,95	1	SÍ
7	7	8	7	8	10	35,75	0,32	SÍ
6	8	8	7	8	9	37,25	0,08	NO
5	6	8	8	8	10	36,8	0,12	NO
4	7	8	6	9	10	34,2	1	SÍ
3	6	8	6	9	11	33	1	SÍ
2	6	8	7	9	10	34,9	0,000075	NO
1	6	8	6	10	10	33,45	0,01	NO
Final	6	8	6	9	11	33		

A,B,C,D,E són els cinc servidors de l'exemple

Com es pot observar, és més probable que canvis d'estat que incrementin l'energia (el temps mitjà d'espera) s'acceptin al principi de l'execució (per exemple, en la iteració 7) que cap al final (iteració 1). En aquest exemple d'execució el millor estat és l'últim que s'ha trobat. Recordem que no necessàriament ha de ser així, ja que l'algorisme emmagatzema el millor estat trobat i és possible (encara que no gaire probable) que l'estat final sigui pitjor.

Un últim aclariment: atesa la seva simplicitat, aquest exemple es podria haver resolt de manera analítica, i obtenir-ne la solució òptima fàcilment. En aquest cas l'ús de recocció simulada no tindria sentit, encara que s'ha utilitzat per a presentar un exemple suficientment senzill. En problemes reals més complexos i amb més variables en joc, les solucions analítiques resulten, en general, impossibles, i llavors sorgeix la necessitat d'utilitzar mètodes com la recocció simulada.

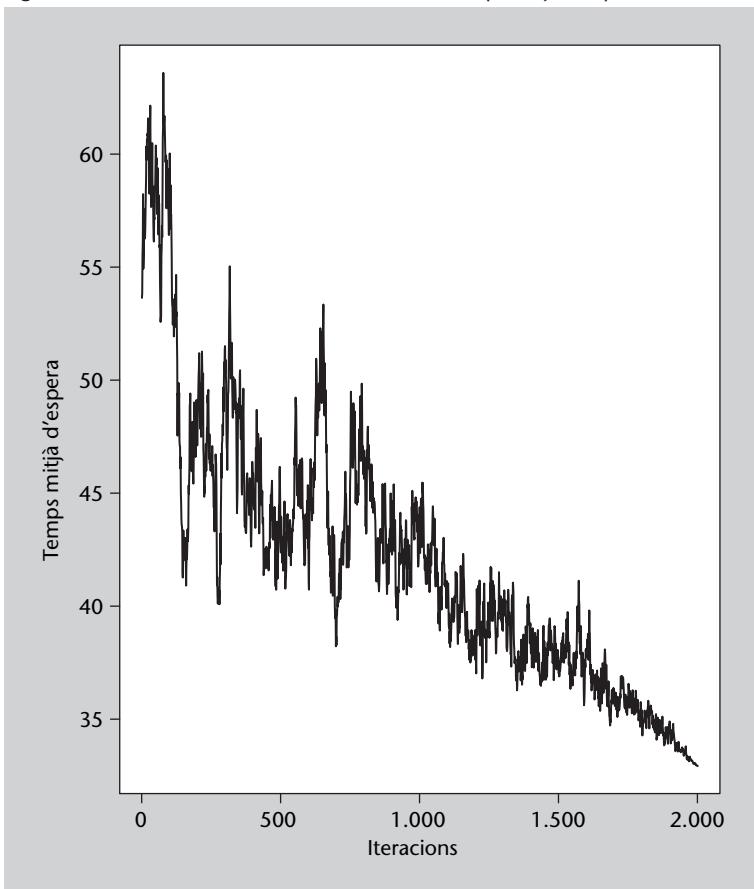
5.3.3. Anàlisi del mètode

La recocció simulada és aplicable a pràcticament qualsevol problema d'optimització, i un dels seus principals avantatges és que no requereix triar gaires paràmetres de funcionament, a diferència d'altres mètodes d'optimització.

La seva aplicació és més evident en problemes l'espai de cerca dels quals és discret, ja que d'aquesta manera la definició d'estat veí és més directa; quan l'espai de cerca és continu, és necessari establir una distància de veïnatge que introduceix un paràmetre més en el mètode.

En la figura 51 es pot veure com descendeix el temps mitjà d'espera en funció del nombre d'iteracions. En aquest cas hi ha 50 servidors i 400 clients. Noteu com en les primeres iteracions el temps mitjà d'espera puja i baixa, ja que l'algorisme té una temperatura més alta. A mesura que el nombre d'iteracions creix, el temps mitjà disminueix o augmenta molt lleugerament, ja que la temperatura és més baixa i no s'accepten canvis d'estat que pugin molt l'energia.

Figura 51. Efecte del nombre d'iteracions en el temps mitjà d'espera



D'altra banda, en la figura 52 es compara el rendiment de la recocció aleatòria enfront de la cerca aleatòria. Com es veu, en les primeres iteracions la cerca

aleatòria obté resultats millors. Això és a causa que en les primeres iteracions la recocció tolera canvis a estats pitjors, tal com s'ha explicat anteriorment. No obstant això, a mesura que creix el nombre d'iteracions, la cerca aleatòria es tendeix a estancar, mentre que la recocció continua millorant els resultats obtinguts, en aquest cas reduint el temps mitjà d'espera. Noteu també que el nombre d'iteracions que requereix la recocció simulada és relativament alt, ja que a cada moment només s'explora una solució possible.

Figura 52. Recocció simulada enfront de cerca aleatòria

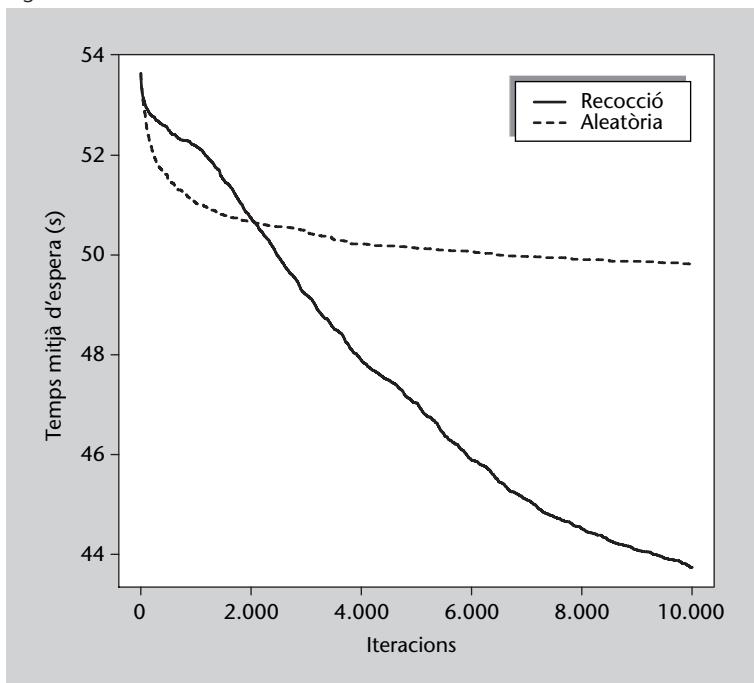


Figura 52

Comparativa entre recocció simulada i cerca aleatòria. Es mostra el temps mitjà d'espera obtingut en un problema amb 400 clients i 50 servidors. Els valors mostrats són la mitjana de 100 execucions diferents.

5.3.4. Codi font en Python

Codi 5.1: Algorisme de recocció simulada

```

1 import random, sys, math, operator
2
3 # Calcula el temps mitjà d'espera d'un conjunt de
4 # servidors i una distribució de clients
5 def tempsMitjaEspera(servidores, clients):
6     temps = [t*(1+c) for (t,c) in zip(servidores, clients)]
7     suma   = sum(map(operator.mul, temps, clients))
8     return suma / float(sum(clients))
9
10
11 # Genera un veí de l'estat actual: simplement mou un
12 # client d'un servidor a un altre a l'atzar
13 def generaVei(estat):
14     nou = estat [:]
15     fet = False
16     while not fet:
17         triats = random.sample(range(len(estado)), 2)
18         origen  = triats[0]
19         destinacio  = triats[1]
20
21         # El nombre de clients en un servidor ha de ser >=0

```

```

22     if nou[origen] > 0:
23         nou[origen] -= 1
24         nou[destinacio] += 1
25         fet = True
26
27     return nou
28
29
30 # Tenint en compte l'energia de cada estat, la iteració
31 # actual i el factor de tolerància, decideix si un nou estat
32 # s'accepta o no
33 def accepta(energia, novaEnergia, iteracions, factor):
34     if novaEnergia < energia:
35         return True
36     else:
37         valor = math.exp((energia-novaEnergia)/
38                           (iteracions*factor))
39     return random.random() < valor
40
41
42 # Aplica l'algoritme de recocció simulada a un conjunt de
43 # servidors i una distribució de clients. La tolerància indica
44 # l'empitjorament acceptable en iniciar l'algoritme (aproximat).
45 def recoccioSimulada(servidors, clients, tolerancia,
46                      iteracions):
47
48     # La tolerància permet ajustar la temperatura per controlar
49     # si s'accepta o no un nou estat "pitjor" (amb més temps
50     # de resposta).
51     factor = tolerancia / float(iteracions)
52
53     # Es calcula el temps mitjà d'espera de l'estat inicial
54     estat = clients [:]
55     temps = tempsMitjaEspera(servidors, estat)
56
57     # S'emmagatzema el millor estat obtingut; serà el que es retornarà
58     millor = estat [:]
59     millorTemps = temps
60     for i in range(iteracions):
61         nou = generaVei(estat)
62         nouTemps = tempsMitjaEspera(servidors, nou)
63
64         if accepta(temps, nouTemps, iteracions, factor):
65             estat = nou
66             temps = nouTemps
67
68         if nouTemps < millorTemps:
69             millor = estat [:]
70             millorTemps = temps
71
72     return millor
73
74
75 # Paràmetres generals del problema
76 # Tolerància indica aproximadament el màxim empitjorament
77 # que s'accepta inicialment
78 numServidors = 50
79 numClients = 400
80 maxTResposta = 10
81 tolerancia = 1.0
82 iteracions = 2000
83
84 # Generació d'un conjunt de servidors
85 servidors = [random.randint(1,maxTResposta)]
86             for i in range(numServidors)]
87 print(servidors)
88
89 # Inicialment els clients es distribuixen per igual entre
90 # tots els servidors; si en sobra cap, s'assigna a l'atzar
91 clients = [numClients/numServidors]
92                     for i in range(numServidors)]
```

```

94 resta = numClients - sum(clients)
95 if resta >0:
96     for serv in random.sample(range(numServidores), resta):
97         clients[serv] += 1
98
99 print(clients, tempsMitjaEspera(servidores, clients))
100
101 # Es crida recacció simulada perquè obtingui una bona
102 # distribució dels clients en els servidors
103 clients = recoccioSimulada(servidores, clients, tolerancia,
104                               iteracions)
105
106 print(clients, tempsMitjaEspera(servidores, clients))

```

5.4. Algorismes genètics

Suposem que una empresa d'inversions en línia ens encarrega que construïm un assessor d'inversions per als seus clients. El plantejament del problema és el següent: l'empresa ofereix als seus clients la possibilitat d'invertir en bons de diverses companyies. Els bons de cada companyia tenen tres característiques: preu per bo, rendiment dels bons (suposarem que és conegut) i quantitat de bons disponibles. El client, per la seva banda, disposa d'un capital determinat. L'objectiu del nostre assessor d'inversions és que el client obtingui el rendiment màxim del seu capital seleccionant la millor combinació de bons. En aquest subapartat estudiarem un potent mètode d'optimització que podem utilitzar per a resoldre aquest problema i molts altres tipus de problemes.

Els éssers vius exhibeixen una enorme diversitat i una sorprendent adaptació als entorns més diversos. Això és possible mitjançant els mecanismes de la **selecció natural**, descrits per Darwin al segle XIX. La idea central de la seva **teoria de l'evolució** és que els organismes més ben adaptats al medi són els que tenen més possibilitats de sobreviure i deixar descendència, i per tant són més abundants que els més mal adaptats, que possiblement s'acabaran extingint.

Les característiques d'un organisme estan determinades per la seva **informació genètica**, que s'hereta de pares a fills. D'aquesta manera, les característiques que ajuden a la supervivència són les que s'hereten amb més probabilitat. En el cas de la reproducció sexual, la informació genètica del pare i la de la mare es combinen de manera aleatòria per a formar la informació genètica dels fills. És el que es diu l'**encreuament**. Es tracta d'un procés molt complex, atès el gran volum d'informació transferit, per la qual cosa poc sovint es produeix un error de còpia, que dóna lloc a una **mutació**, que d'ara endavant heretaran els descendents del portador.

La informació genètica d'un individu es divideix en **gens**, que són seqüències d'informació genètica que descriuen una característica de l'individu*.

*Tingueu en compte que es tracta d'una descripció molt simplificada. La realitat és bastant més complexa.

Els **algorismes genètics** emulen la selecció natural sobre un conjunt d'individus per a buscar la millor solució a un problema determinat. La “informació genètica” de cada individu és una possible solució al problema; per analogia, hi ha un “gen” per a cada variable o paràmetre del problema sobre el qual es vol executar el procés d’optimització. Per a emular la selecció natural, es crea una **població** o conjunt d’individus i es fa evolucionar de manera que els més ben adaptats, o sigui, els que són solució millor per al problema, es reproduueixin amb més probabilitat i a poc a poc vagin sorgint individus més ben adaptats al problema; en altres paraules, solucions millors.

5.4.1. Descripció del mètode

Sigui Q el problema que volem resoldre, i S l’espai de les solucions de Q , en el qual cada solució $s \in S$ es compon de n variables de la forma $s = \{x_1, x_2, x_3, \dots, x_n\}$. Aquestes variables poden ser de qualsevol tipus de dades; les més habituals en aquest cas són binàries, enters i reals, o una combinació d'aquests. A més, cada variable té un rang de valors propi. En un algorisme genètic, cada individu és una solució $s_i \in S$ amb n variables (o gens, si seguim la metàfora biològica).

També hi ha una funció $f : S \rightarrow \mathbb{R}$ que mesura la *idoneïtat** d'una solució $s \in S$, i que en definitiva és la funció que volem maximitzar. En l’àmbit de la selecció natural, la idoneïtat d'un individu correspon a la seva adaptació al medi; els més ben adaptats sobreviuran amb més probabilitat.

*En anglès *fitness*

El punt de partida d'un algorisme genètic és una població P formada per k individus presos a l’atzar de S . A continuació, s’avalua la idoneïtat de cada solució $s \in P$ utilitzant f , i els individus s’ordenen de menor a major idoneïtat (creixent numèricament si es desitja maximitzar f , decreixent si es desitja minimitzar-la).

Observació

Dependent del problema i de la definició de f , l’objectiu pot ser *minimitzar* aquesta funció. No obstant això, en aquest subapartat suposarem que es vol maximitzar f .

Cada iteració d'un algorisme genètic rep el nom de **generació**, novament per analogia amb els organismes vius. En cada generació es pren la població existent, ordenada per idoneïtat, i es genera una nova població mitjançant **l’increuament** dels individus existents de dos en dos. Per a emular la selecció natural, els individus més idonis es reproduueixen amb més probabilitat que els menys idonis; d'aquesta manera és més probable que els seus gens es transmetin a la generació següent d’individus.

Hi ha diferents mètodes per a enkreuar els gens de dos individus A i B : tallar per un gen determinat de A i inserir els gens de B a partir d'aquest punt (increuament en un punt, vegeu la figura 53); o tallar per dos punts de A i inserir els gens de B corresponents (increuament en dos punts, que és el mètode que usarem en el nostre exemple, vegeu la figura 54); també és possible enkreuar a l’atzar cadascun dels gens (increuament uniforme, vegeu la figura 55), en-

tre altres mètodes. Amb freqüència el mètode d'encreuament ha de tenir en compte la naturalesa del problema, ja que pot ocórrer que sobre els gens operin algunes restriccions: hagin d'estar ordenats, no s'hagin de repetir, la seva suma no hagi d'excedir una certa quantitat, etc. L'objectiu final de l'encreuament és, emulant la naturalesa, combinar les característiques avantatjoses de dos individus per a aconseguir un individu més ben adaptat fins i tot al seu entorn.

Figura 53. Encreuament en un punt

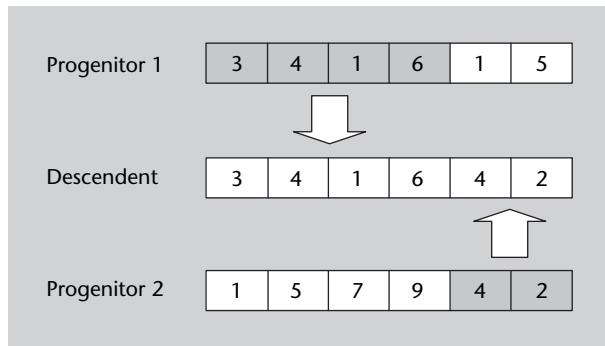


Figura 54. Encreuament en dos punts

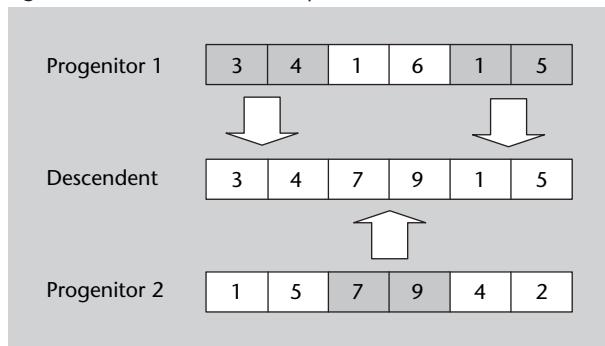
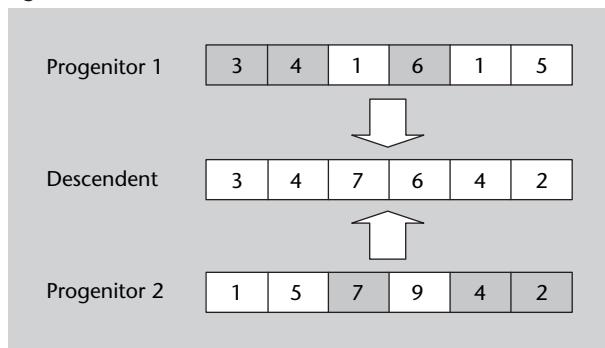


Figura 55. Encreuament uniforme



L'altre mecanisme que intervé en l'evolució biològica, i que els algorismes genètics també utilitzen, és el de la **mutació**. Els individus que s'han generat per encreuament poden sofrir, amb una determinada probabilitat, un canvi aleatori en algun dels seus gens. L'objectiu de les mutacions és enriquir la població introduint gens que no s'hi trobin, la qual cosa permet al seu torn explorar noves zones de l'espai de solucions S , que poden contenir solucions

millors. La probabilitat o taxa de mutació no sol ser gaire alta. En l'exemple presentat a continuació és d'un 1%, a manera d'orientació.

Seguidament s'avalua la idoneïtat d'aquesta nova població, que al seu torn donarà naixement a una nova generació d'individus, i així successivament. No hi ha un criteri únic per a establir el nombre de generacions (iteracions) d'un algorisme genètic; pot ser fix, o fins a assolir una certa idoneïtat, o fins que els nous individus no aportin cap millora durant algunes generacions.

5.4.2. Ampliacions i millores

Hi ha nombroses variants i millores dels algorismes genètics. Una de les més habituals és la de l'**elitisme**: en cada generació, els E individus millors passen inalterats a la generació següent. D'aquesta manera l'algorisme no perd solucions que poden ser millors que la solució final, i a més permet provar nombrosos encreuaments i mutacions de les millors solucions. Freqüentment es pren $|E| = 1$, com en l'exemple desenvolupat més endavant.

Altres propostes per a millorar els algorismes genètics inclouen l'encreuament entre més de dos individus i els algorismes **memètics**, en els quals els individus tenen la capacitat d'aprendre i transmetre l'aprés als seus descendents.

5.4.3. Exemple d'aplicació

A continuació dissenyarem i aplicarem un algorisme genètic al problema exposat abans de l'assessor d'inversions. En realitat es tracta d'una de les moltes variants del conegut **problema de la motxilla**.

Per fer-nos una idea de la complexitat d'aquest problema, suposem que de cada tipus de bo hi hagués 10 unitats disponibles. Si tinguéssim 100 tipus de bons diferents, el nombre de combinacions possibles d'inversió seria 10^{100} . Evidentment, és impossible intentar una exploració exhaustiva de totes les combinacions, per la qual cosa es necessita un algorisme que trobi una combinació raonablement bona en un temps acceptable.

En la taula 36 es planteja un exemple de cinc tipus de bons, en el qual cada tipus té un preu, quantitat i rendiment diferent. En la taula 37 es pot seguir l'execució de l'algorisme durant dues generacions, i de quina manera operen la selecció dels millors individus, el seu encreuament (en un punt) i les mutacions. A més, en aquest algorisme es manté sempre el millor individu de cada generació (elitisme). L'últim individu de la taula és el que proporciona més bon rendiment de tots els obtinguts per l'algorisme.

Taula 36. Tipus de bons disponibles

	A	B	C	D	E
Preu	500	700	300	800	400
Quantitat	5	6	10	3	7
Rendiment	0,15	0,05	0,03	0,1	0,1

Taula 37. Execució de l'assessor d'inversions

Individu	A	B	C	D	E	Capital	Rendiment
Generació inicial							
1	1	1	2	3	1	4600	408
2	1	2	0	2	3	4700	425
3	4	3	1	0	1	4800	454
4	2	0	1	3	3	4900	519
Generació 1							
5 (4+2)	2	2	0	2	3	4500	465
6 (4+3)	4	0	1	2	2	4700	549
7 (3+2)	1	2	1	2	3	5000	434
4 (Elit)	2	0	1	3	3	4900	519
Mutació							
7 (3+2)	1	1	1	2	3	4300	399
5 (4+2)	2	1	0	2	3	4500	465
4 (Elit)	2	0	1	3	3	4900	519
6 (4+3)	4	0	1	2	2	4700	558
Generació 2							
8 (6+7)	4	0	1	2	3	4800	580
9 (6+5)	2	1	0	2	2	4100	425
10 (6+4)	3	0	1	2	4	4600	514
6 (Elit)	4	0	1	2	2	4700	558
Mutació							
9 (6+5)	2	1	1	2	2	4400	434
10 (6+4)	3	0	1	2	4	5000	554
6 (Elit)	4	0	1	2	2	4700	558
8 (6+7)	4	0	0	2	3	4800	580

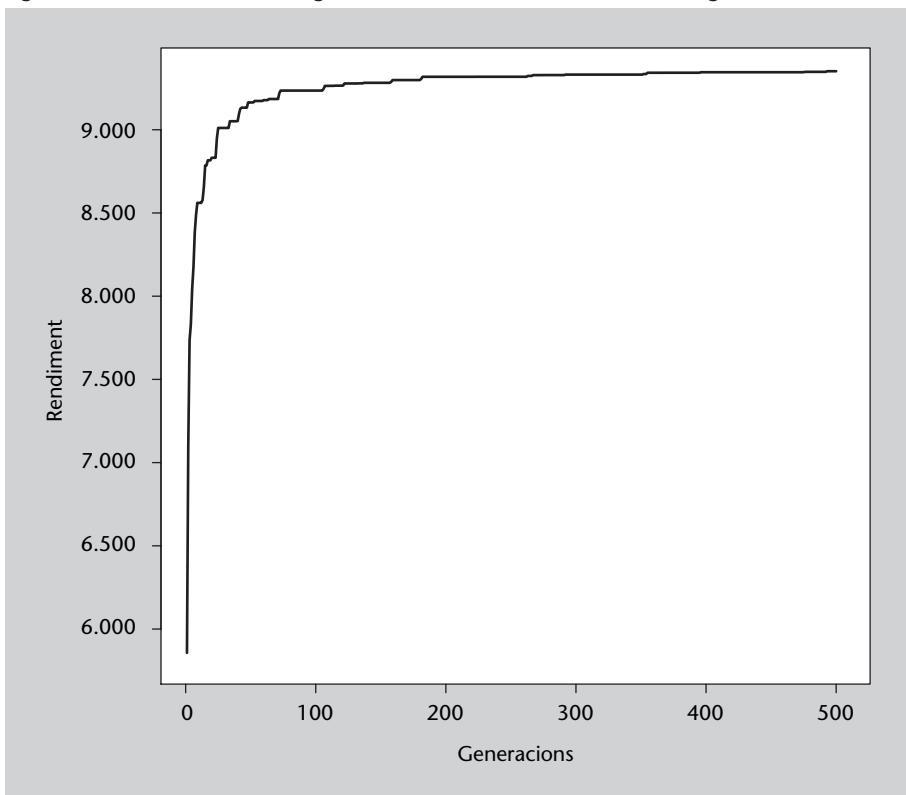
Algunes combinacions poden excedir el capital disponible; en tal cas, s'eliminen algunes inversions a l'atzar (valors ratllats).

5.4.4. Anàlisi del mètode

Els algorismes genètics són aplicables a gran nombre de problemes d'optimització, i en general troben solucions raonablement bones en un temps acceptable. Són, per tant, una opció per considerar sempre que es plantegi un problema d'optimització.

En la figura 56 es mostra el rendiment obtingut en el problema de les inversions en línia, amb 100 tipus de bons i 20 individus.

Figura 56. Efecte del nombre de generacions en el rendiment màxim obtingut



No obstant això, tenen alguns problemes. El principal és que depenen d'un gran nombre de paràmetres i operacions (nombre d'individus, taxa de mutació, estratègia d'encreuament) i no hi ha regles que indiquin clarament quins valors hem de triar per a cada problema. Per tant, cal ajustar aquests paràmetres seguit recomanacions molt generals.

Una altra crítica que reben els algorismes genètics és que, en proposar un gran nombre de solucions al problema, es requereix calcular la funció d'idoneïtat moltes vegades; sovint aquesta funció és molt costosa en temps de computació, la qual cosa fa que utilitzar un algorisme genètic requereixi molt temps d'execució.

5.4.5. Codi font en Python

Codi 5.2: Algorisme genètic de l'assessor d'inversions

```

1  from random import random, randint, sample
2  from collections import namedtuple
3
4  # Calcula el capital invertit per un individu
5  def capitalInvertit(individu):
6      return sum(map(lambda x,y: x*y.preu, individu, inversions))
7

```

```

8 # Calcula el rendiment obtingut per un individu
9 def rendiment(individu):
10    return sum(map(lambda x,y: x*y.preu*y.rendim,
11                  individu, inversions))
12
13
14 # Si un individu gasta més capital del disponible, s'eliminen
15 # aleatòriament inversions fins que s'ajusta al capital
16 def ajustaCapital(individu):
17    ajustat = individu[:]
18    while capitalInvertit(ajustat)>capital:
19        pos = randint(0,len(ajustat)-1)
20        if ajustat[pos] > 0:
21            ajustat[pos] -= 1
22
23    return ajustat
24
25
26 # Crea un individu a l'atzar, en aquest cas una selecció
27 # d'inversions que no excedeixin el capital disponible
28 def creaIndividu(inversions, capital):
29    individu = [0]*len(inversions)
30
31    while capitalInvertit(individu) < capital:
32        eleccio = randint(0,len(inversions)-1)
33        individu[eleccio] += 1
34
35    return ajustaCapital(individu)
36
37
38 # Crea un nou individu encreuant-ne dos més (les posicions dels quals
39 # s'indiquen en el segon paràmetre)
40 def encreua(poblacio, posicions):
41    L = len(poblacio[0])
42
43    # Pren els gens del primer progenitor i després pren a l'atzar
44    # un segment d'entre 1 i L gens del segon progenitor
45    fill = poblacio[posicions[0]][:]
46    inici = randint(0,L-1)
47    fi = randint(inici+1,L)
48    fill[inici:fi] = poblacio[posicions[1]][inici:fi]
49
50    return ajustaCapital(fill)
51
52
53 # Aplica mutacions a un individu segons una taxa donada; garanteix
54 # que compleix les restriccions de capital i inversions
55 def muta(individu, taxaMutació):
56    mutat = []
57    for i in range(len(individu)):
58        if random() > taxaMutació:
59            mutat.append(individu[i])
60        else:
61            mutat.append(randint(0,inversions[i].quantitat))
62
63    return ajustaCapital(mutat)
64
65
66 # Fa evolucionar el sistema durant un nombre de generacions
67 def evoluciona(poblacio, generacions):
68
69    # Ordena la població inicial per rendiment produït
70    poblacio.sort(key=lambda x:rendiment(x))
71
72    # Alguns valors útils
73    N = len(poblacio)
74    taxaMutació = 0.01
75
76    # Genera una llista del tipus [0,1,1,2,2,2,3,3,3,...] per
77    # representar les probabilitats de reproduir-se de cada
78    # individu (el primer 1 possibilitat, el segon 2, etc.)
79    reproduccio = [x for x in range(N) for i in range(x+1)]

```

```

80
81     for i in range(generacions):
82         # Es generen N-1 nous individus encreuant els existents
83         # (sense que es repeteixin els pares)
84         pares = sample(reproduccio,2)
85         while pares[0]==parees[1]:
86             pares = sample(reproduccio,2)
87             fills = [encreua(poblacio, pares) for x in range(N-1)]
88
89         # S'apliquen mutacions amb una certa probabilitat
90         fills = [muta(x, taxaMutacio) for x in fills]
91
92         # 'afegeix el millor individu de la població anterior
93         # (elitisme)
94         fills.append(poblacio[-1])
95         poblacio = fills
96
97         # S'ordenen els individus per rendiment
98         poblacio.sort(key=lambda x:rendiment(x))
99
100
101    # Retorna el millor individu trobat
102    return poblacio[-1]
103
104
105 # Declara un tuple amb noms per representar cada inversió
106 Inversió = namedtuple('Inversio', 'preu_quantitat_rendim')
107
108 numInver = 100
109 maxPreu = 1000
110 maxQuant = 10
111 maxRend = 0.2
112
113
114 # Genera una llista de tuples Inversió
115 inversions=[Inversio(random()*maxPreu, randint(1,maxQuant),
116                         random()*maxRend) for i in range(numInver)]
117 print (inversions)
118
119 capital = 50000
120 individus = 20
121 generacions = 1000
122
123 poblacio = [creaIndividu(inversions, capital)
124                         for i in range(individus)]
125
126 # Nota: per a simplificar el programa s'accedeix a inversions i
127 # capital de forma global (només es llegeixen, no es modifiquen)
128
129 millor = evoluciona(poblacio, generacions)
130 print(millor, capitalInvertit(millor), rendiment(millor))

```

5.5. Colònies de formigues

El **problema del viatjant de comerç** és un problema clàssic d'optimització en el qual es disposa d'un conjunt de ciutats, amb una distància entre cada parell de ciutats. El viatjant al qual es refereix el problema ha de visitar totes les ciutats una i només una vegada, intentant per la seva banda recórrer la mínima distància (per gastar menys temps, combustible, etc.).

L'algorisme d'optimització mitjançant colònies de formigues* és un algorisme probabilístic que imita l'habilitat de les formigues per a trobar el camí més

* En anglès, *ant colony optimization*.

curt des del seu formiguer fins a una font d'aliment. Les formigues són un excel·lent exemple de sistema emergent, atès que cada formiga individual manca de la intel·ligència suficient per a trobar el camí més curt fins a una font d'aliment, i no obstant això el seu comportament coordinat ho aconsegueix.

La manera en què les formigues aconsegueixen trobar el camí més curt és la següent: en principi les formigues vagaregen a l'atzar al voltant del seu formiguer, i quan troben aliment en prenen una mica i hi tornen deixant un rastre de feromones. Si altres formigues troben aquest rastre, és probable que deixin de voltar a l'atzar i el segueixin, ja que suposadament condueix a una font d'aliment. Al seu torn, les formigues que tornen amb aliment deixen el seu rastre propi de feromona, i es reforça així aquest camí. Si hi ha diversos camins cap a una mateixa font d'aliment el més curt acabarà essent el preferit per les formigues, per la senzilla raó que en ser més curt el recorren més formigues per unitat de temps, i per tant la intensitat del seu rastre de feromones serà més gran. D'altra banda, les feromones s'evaporen gradualment, amb la qual cosa els camins que no s'utilitzen van perdent atractiu. D'aquesta manera, partint d'una exploració aleatòria s'aconsegueix trobar un camí òptim o proper a l'òptim.

5.5.1. Descripció del mètode

L'adaptació d'aquest comportament a un algorisme d'optimització és relativament directa*: es genera una "formiga" que en principi explora l'espai del problema de manera aleatòria, i quan troba una solució al problema, la marca positivament amb "feromones", de manera que les formigues següents seguiran aquest camí amb una probabilitat més alta, si bé mantenint un cert component aleatori.

L'estrucció general de l'algorisme és molt senzilla: en cada iteració es genera una "formiga", que proposa una solució al problema. El criteri de finalització pot ser tan senzill com assolir un nombre d'iteracions predeterminat, o pot ser més elaborat, com assolir una solució de certa qualitat o comprovar que successives iteracions no milloren la solució obtinguda.

Les solucions parcials del problema es denominen **estats de la solució**. Per a generar una solució completa cal partir de l'estat inicial i anar movent-se a un nou estat de la solució fins a completar la solució; la naturalesa específica de cada problema determina quina és una solució completa en cada cas. A cada moment, la probabilitat P_{xy} que una formiga passi d'un estat x a un estat y , depèn de dos factors: la facilitat ϕ_{xy} per a passar de l'estat x al y , i el rastre de

Sistemes emergents

Els sistemes emergents són aquells que resolen problemes complexos i exhibeixen un elevat grau d'intel·ligència combinant subsistemes senzills i poc intel·ligents. Les neurones són potser l'exemple més clar.

* Va ser proposada per primera vegada per M. Dorigo en la seva tesi doctoral *Optimization, Learning and Natural Algorithms*, Politecnico di Milano, Itàlia, 1992.

feromones ρ_{xy} que en aquest moment està associat a aquesta transició. Si E és el conjunt d'estats de la solució, llavors la probabilitat de triar un nou estat està determinada per

$$P_{xy} = \frac{(1 + \rho_{xy}^\alpha) \phi_{xy}^\beta}{\sum_{i \in E - \{x\}} (1 + \rho_{xi}^\alpha) \phi_{xi}^\beta} \quad (57)$$

en què el paràmetre $\alpha \geq 0$ controla la influència del rastre de feromones ρ_{xy} i el paràmetre $0 \leq \beta \leq 1$ controla la influència de la facilitat de transició ϕ_{xy} . En altres paraules, la probabilitat de canviar a un estat y es calcula comparant la facilitat i nivell de feromones del camí xy amb les de la resta de camins disponibles des de x .

Cada vegada que una formiga completa una solució és necessari actualitzar el rastre de feromones entre estats de la solució. S'apliquen dues operacions: d'una banda, les feromones es tendeixen a evaporar amb el temps; d'una altra, la formiga deixa un rastre de feromones pel camí que ha seguit. Així, la quantitat de feromona associada a la transició entre els estats x i y està determinada per:

$$\rho_{xy} = (1 - \gamma)\rho_{xy} + \Delta\rho_{xy} \quad (58)$$

en què $0 \leq \gamma \leq 1$ és el coeficient d'evaporació de les feromones i $\Delta\rho_{xy}$ és el rastre deixat per la formiga en la transició de x a y , típicament zero quan la formiga no ha utilitzat aquesta transició en la seva solució, i un valor constant quan la transició sí que forma part de la solució.

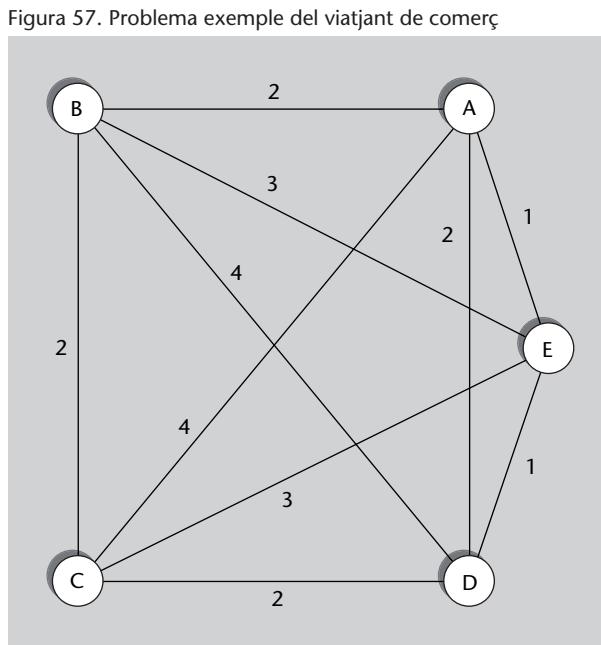
Una millora habitual d'aquest algorisme consisteix a permetre que les formigues deixin un rastre de feromones només quan han trobat un camí millor que el millor camí fins a aquest moment, de manera que es premiï la troballa de camins millors i es redueixi el nombre d'iteracions necessari per a trobar-los.

5.5.2. Exemple d'aplicació

Les aplicacions més evidents d'aquest algorisme són aquelles en les quals es busca el camí més curt entre dos punts, si bé s'han aplicat amb èxit a tasques tan diverses com planificació, classificació, partició de conjunts, processament d'imatges i plegament de proteïnes.

Com a exemple d'aplicació de l'optimització amb colònies de formigues resoldrem el problema del viatjant de comerç descrit anteriorment. Podem formalitzar el problema representant-lo com un graf en el qual cada ciutat és un node i els arcs estan etiquetats amb la distància d'una ciutat a una altra. En

aquest exemple anem a suposar que la ciutat d'origen és la A, i que el viatjant hi ha de tornar al final del seu trajecte. En la figura 57 podem veure un exemple d'aquest problema, en el qual els valors dels arcs representen les distàncies entre ciutats.



Per a resoldre aquest problema mitjançant l'algorisme de la colònia de formigues generarem successives formigues. La primera recorrerà les ciutats en un ordre aleatori, si bé serà més probable que viatgi a les ciutats més properes, una restricció lògica si intentem simular el comportament d'una formiga real. Així, la facilitat d'anar d'un node x a un node y serà $\phi_{xy} = 1/dist(x,y)$. Una solució d'aquest problema serà una seqüència de ciutats en la qual cada ciutat aparegui una i només una vegada. La solució òptima serà aquella que minimitzi la distància total recorreguda per la formiga, incloent-hi la volta a la ciutat de partida després de recórrer les ciutats restants.

El criteri de finalització emprat en aquest exemple és molt senzill: un nombre d'iteracions (formigues) predeterminat, exactament 3, per no allargar excessivament l'exemple. Tal com s'ha explicat anteriorment, és possible utilitzar un criteri més sofisticat. L'exemple d'execució no pretén mostrar els avantatges de l'algorisme, ja que fer-ho requeriria resoldre un problema amb gran nombre de ciutats i d'iteracions, sinó simplement il·lustrar el funcionament bàsic de l'algorisme; la seva potència veritable s'aprecia en problemes més importants.

Execució de l'algorisme

La taula 38 mostra el pes inicial de les transicions entre els estats (ciutats) del problema proposat en la figura 57. Es pot observar que, inicialment, no hi ha rastres de feromones; per tant, la probabilitat que una formiga prengui una

transició determinada és inversament proporcional a la distància a aquesta ciutat, tal com s'ha proposat abans. Noteu també que en aquest exemple no es distingeixen dos sentits en les transicions entre una ciutat i una altra, i aquesta és la raó per la qual només es mostra la meitat superior de la taula.

Taula 38. Estat inicial del problema del viatjant

	B	C	D	E
A	$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 0) \frac{1}{4} = 0,25$	$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 0)1 = 1$
B		$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 0) \frac{1}{4} = 0,25$	$(1 + 0) \frac{1}{3} = 0,33$
C			$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 0) \frac{1}{3} = 0,33$
D				$(1 + 0)1 = 1$

Valor $(1 + \rho_{xy})\phi_{xy}$ per a anar de la ciutat x a la ciutat y

En la primera iteració de l'algorisme, una primera formiga recorre les ciutats en un ordre aleatori, si bé condicionat per les probabilitats de cada transició.

Per tant, aquesta primera formiga executa un algorisme àvid aleatori, en el qual tria el camí aleatoriament, si bé les transicions més atractives a cada moment es trien amb més probabilitat, encara que a llarg termini es tracti de males decisions.

La ruta seguida per la formiga es marcarà amb feromones, fent que les formigues següents segueixin aquestes transicions amb més probabilitat. En aquest cas la ruta seguida per la formiga ha estat $A \rightarrow E \rightarrow D \rightarrow B \rightarrow C \rightarrow A$, amb una longitud total de $1 + 1 + 4 + 2 + 4 = 12$.

D'altra banda, en acabar la primera iteració les feromones sofreixen una certa evaporació d'acord amb l'equació 58. En aquest exemple prendrem $\gamma = 0,3$, un valor relativament alt atès que en aquest exemple el nombre d'iteracions és molt reduït; per regla general convé prendre un valor més petit, com per exemple 0,1. Tota aquesta informació es mostra en la taula 39.

Taula 39. Problema del viatjant després de la primera iteració

	B	C	D	E
A	$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 0,7) \frac{1}{4} = 0,425$	$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 0,7)1 = 1,7$
B		$(1 + 0,7) \frac{1}{2} = 0,85$	$(1 + 0,7) \frac{1}{4} = 0,425$	$(1 + 0) \frac{1}{3} = 0,33$
C			$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 0) \frac{1}{3} = 0,33$
D				$(1 + 0,7)1 = 1,7$

D'aquesta manera, la formiga següent es troba unes probabilitats diferents de prendre cada transició, guiada pel recorregut de la formiga anterior. La segona formiga segueix el camí $A \rightarrow E \rightarrow C \rightarrow D \rightarrow B \rightarrow A$, amb una longitud de $1 + 3 + 2 + 4 + 2 = 12$; en no millorar la solució anterior, aquesta formiga no deixa feromones, encara que les feromones existents sofreixen una certa evaporació. En la taula 40 es pot veure l'estat de les transicions després de la segona iteració.

Taula 40. Problema del viatjant després de la segona iteració

	B	C	D	E
A	$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 0,49) \frac{1}{4} = 0,373$	$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 0,49)1 = 1,49$
B		$(1 + 0,49) \frac{1}{2} = 0,745$	$(1 + 0,49) \frac{1}{4} = 0,373$	$(1 + 0) \frac{1}{3} = 0,33$
C			$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 0) \frac{1}{3} = 0,33$
D				$(1 + 0,49)1 = 1,49$

L'última formiga segueix el camí $A \rightarrow E \rightarrow D \rightarrow C \rightarrow B \rightarrow A$, amb una longitud de $1 + 1 + 2 + 2 + 2 = 8$, més curt que el millor camí trobat fins ara; de fet, és fàcil comprovar que és el camí òptim per a aquest problema (com el camí invers, obviament). En aquest cas la formiga sí que deixa un rastre de feromones, que seria utilitzat per les formigues següents, si n'hi ha. En la taula 41 es pot veure l'estat final de les transicions.

Taula 41. Problema del viatjant després de la tercera iteració

	B	C	D	E
A	$(1 + 0,7) \frac{1}{2} = 0,85$	$(1 + 0,49) \frac{1}{4} = 0,373$	$(1 + 0) \frac{1}{2} = 0,5$	$(1 + 1,043)1 = 2,043$
B		$(1 + 1,043) \frac{1}{2} = 1,022$	$(1 + 0,49) \frac{1}{4} = 0,373$	$(1 + 0) \frac{1}{3} = 0,33$
C			$(1 + 0,7) \frac{1}{2} = 0,85$	$(1 + 0) \frac{1}{3} = 0,33$
D				$(1 + 1,043)1 = 2,043$

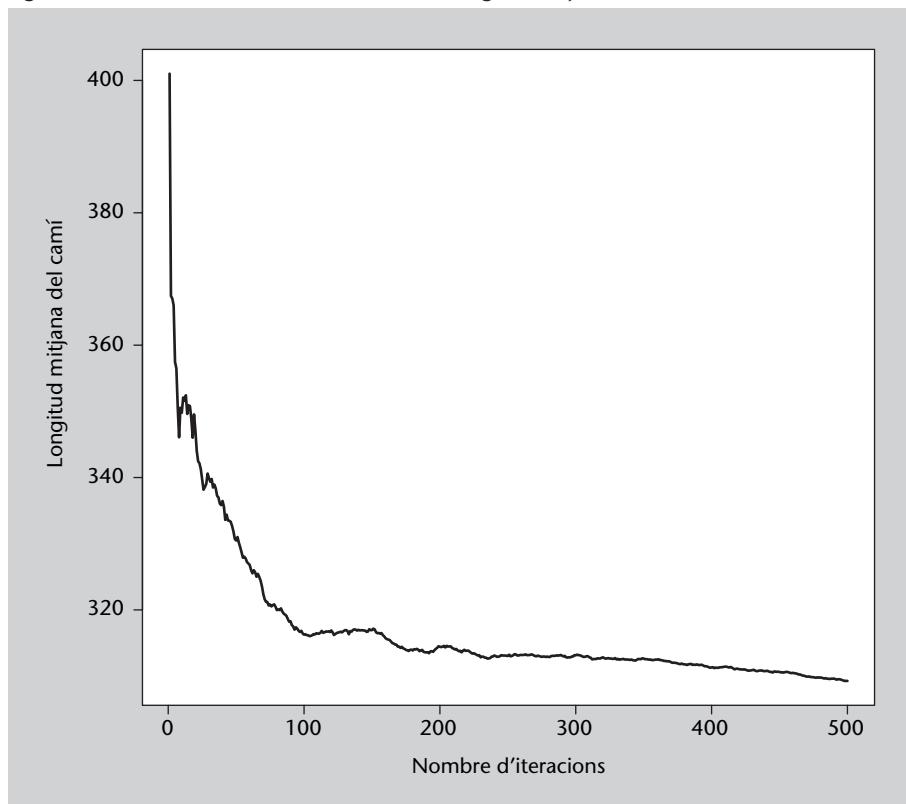
5.5.3. Anàlisi del mètode

Aquest mètode d'optimització té nombrosos avantatges, ja que permet trobar solucions raonablement bones a problemes molt complexos millorant progressivament solucions aleatòries. A més, en general evita els mínims locals. La seva aplicabilitat és bastant general, ja que es pot aplicar a qualsevol problema que es pugui traduir a la recerca d'un camí òptim en un graf. No obstant això, en alguns casos aquesta traducció pot ser difícil o molt costosa.

Potser el principal problema d'aquest algorisme és la gran quantitat de paràmetres que cal triar: pesos α i β en el càcul de probabilitats, coeficient d'evaporació γ , etc.

Per il·lustrar l'efecte del nombre d'iteracions en la longitud del camí obtingut, en la figura 58 es mostra l'evolució de la longitud mitjana dels camins seguits per les formigues enfront del nombre d'iteracions (formigues). En aquest exemple s'han pres 100 ciutats amb una distància màxima entre aquestes de 10. Es pot observar que la longitud dels camins baixa ràpidament i s'estabilitza a mesura que s'incrementen les iteracions. Per a valorar l'eficiència de l'algorisme, cal tenir en compte que el nombre de recorreguts possibles en aquest problema és $n!$, en què n és el nombre de ciutats; així doncs, per a 100 ciutats el nombre de recorreguts possibles és $100!$, o sigui, més de 10^{157} recorreguts possibles. Evidentment, una exploració exhaustiva de tots és impracticable.

Figura 58. Efecte del nombre d'iteracions en la longitud mitjana del camí



5.5.4. Codi font en Python

Codi 5.3: Algorisme de la colònia de formigues

```

1 import random, sys, math
2
3 # Nota: en lloc de matrius s'usen llistes de llistes
4
5 # Genera una matriu de distàncies de nCiutats x nCiutats
6 def matriuDistançies(nCiut, distanciaMaxima):
7     matriu = [[0 for i in range(nCiut)] for j in range(nCiut)]
8
9     for i in range(nCiut):
10        for j in range(i):
11            matriu[i][j] = distanciaMaxima*random.random()
12            matriu[j][i] = matriu[i][j]
13
14    return matriu
15
16 # Escull un pas d'una formiga, tenint en compte les distàncies
17 # i les feromones i descartant les ciutats ja visitades.
18 def escullCiutat(dists, ferom, visitades):
19     # Es calcula la taula de pesos de cada ciutat
20     llistaPesos = []
21     disponibles = []
22     actual      = visitades[-1]
23
24     # Influència de cada valor (alfa: feromones; beta: distàncies)
25     alfa = 1.0
26     beta = 0.5
27
28     # El paràmetre beta (pes de les distàncies) és 0.5, alfa=1.0
29     for i in range(len(dists)):

```

```

30     if i not in visitades:
31         fer = math.pow((1.0 + ferom[actual][i]), alfa)
32         pes = math.pow(1.0/dists[actual][i], beta) * fer
33         disponibles.append(i)
34         listaPesos.append(pes)
35
36 # S'escull aleatòriament una de les ciutats disponibles,
37 # tenint-ne en compte el pes relatiu.
38 valor = random.random() * sum(llistaPesos)
39 acumulat = 0.0
40 i = -1
41 while valor > acumulat:
42     i += 1
43     acumulat += llistaPesos[i]
44
45 return disponibles[i]
46
47
48 # Genera una "formiga", que escollirà un camí tenint en compte
49 # les distàncies i els rastres de feromones. Retorna un tupla
50 # amb el camí i la seva longitud.
51 def escullCami(distancies, feromones):
52     # La ciutat inicial sempre és la 0
53     cami = [0]
54     longCami = 0
55
56     # Triar cada pas segons la distància i les feromones
57     while len(cami) < len(distancies):
58         ciutat = escullCiutat(distancies, feromones, cami)
59         longCami += distancies[cami[-1]][ciutat]
60         cami.append(ciutat)
61
62     # Per acabar, s'ha de tornar a la ciutat d'origen (0)
63     longCami += distancies[cami[-1]][0]
64     cami.append(0)
65
66 return (cami, longCami)
67
68 # Actualitza la matriu de feromones seguint el camí rebut
69 def rastreFeromones(feromones, cami, dosi):
70     for i in range(len(cami) - 1):
71         feromones[cami[i]][cami[i+1]] += dosi
72
73 # Evapora totes les feromones multiplicant-les per una constant
74 # = 0.9 (en altres paraules, el coeficient d'evaporació és 0.1)
75 def evaporaFeromones(feromones):
76     for llista in feromones:
77         for i in range(len(llista)):
78             llista[i] *= 0.9
79
80 # Resol el problema del viatjant de comerç mitjançant
81 # l'algoritme de la colònia de formigues. Rep una matriu de
82 # distàncies i retorna un tupla amb el millor camí que ha
83 # obtingut (llista d'índexs) i la seva longitud
84 def formigues(distancies, iteracions, distMitjana):
85     # Primer es crea una matriu de feromones buida
86     n = len(distancies)
87     feromones = [[0 for i in range(n)] for j in range(n)]
88
89     # El millor camí i la seva longitud (inicialment "infinita")
90     millorCami = []
91     longMillorCami = sys.maxint
92
93     # En cada iteració es genera una formiga, que escull un camí,
94     # i si és millor que el millor que teníem, deixa el seu rastre de
95     # feromones (més intens com més curt sigui el camí)
96     for iter in range(iteracions):
97         (cami, longCami) = escullCami(distancies, feromones)
98
99         if longCami <= longMillorCami:
100             millorCami = cami
101             longMillorCami = longCami

```

```

102
103     rastreFeromones(feromones, cami, distMitjana/longCami)
104
105     # En qualsevol cas, les feromones es van evaporant
106     evaporaFeromones(feromones)
107
108
109     # Es retorna el millor camí que s'haig trobat
110     return (millorCami, longMillorCami)
111
112
113 # Generació d'una matriu de prova
114 numCiutats = 10
115 distanciaMaxima = 10
116 ciutats = matriuDistances(numCiutats, distanciaMaxima)
117
118 # Obtenició del millor camí
119 iteracions = 1000
120 distMitjana = numCiutats*distanciaMaxima/2
121 (cami, longCami) = formigues(ciutats, iteracions, distMitjana)
122 print("Cami: ", cami)
123 print("Longitud del cami: ", longCami)

```

5.6. Optimització amb eixams de partícules

Suposem que ens plantegen buscar el mínim (o màxim) d'una funció matemàtica qualsevol, com per exemple

$$f(x,y) = x^2(4 - 2,1x^2 + \frac{x^4}{3}) + xy + y^2(-4 + 4y^2) \quad (59)$$

dins d'un interval determinat, en aquest cas $[-10, -10], [10, 10]$. Es tracta d'un exemple purament il·lustratiu, ja que f és una funció que es pot derivar fàcilment i, per tant, és possible trobar els mínims de manera analítica. Concretament, hi ha dos mínims globals, en $(0,09, -0,71)$ i en $(-0,09, 0,71)$, en els quals la funció pren el valor de $-1,0316$.

El mètode d'optimització amb eixams de partícules* rep el seu nom pel fet que utilitza un conjunt de solucions candidates que es comporten com a partícules movent-se per un espai: l'espai de solucions. El moviment de les partícules es determina per la seva velocitat, que depèn de la posició del millor punt que ha trobat cadascuna en el seu recorregut i del millor punt trobat globalment per l'eixam.

* En anglès, *particle swarm optimization*.

Val la pena esmentar l'origen d'aquest mètode, ja que curiosament va sorigar a partir d'un treball que simulava el comportament d'entitats socials com ramats d'animals. Analitzant el comportament d'aquesta simulació, i simplificant una mica les regles que la governaven, es va observar que les entitats tendien a buscar el punt mínim de l'espai en el qual es trobaven.

Lectura complementària

El mètode d'optimització amb eixams de partícules es va descriure a J. Kennedy; R. Eberhart (1995). "Particle Swarm Optimization". IEEE International Conference on Neural Networks

5.6.1. Descripció del mètode

Suposeu que volem trobar el punt mínim d'una funció $f : \mathbb{R}^n \rightarrow \mathbb{R}$, en què l'espai de solucions $S \subset \mathbb{R}^n$ està fitat pels punts $\inf, \sup \in \mathbb{R}^n$. L'interval de l'espai de solucions S no ha de ser necessàriament el mateix en totes les dimensions.

Primer pas

El primer pas del mètode d'optimització amb eixams de partícules consisteix a crear un conjunt de partícules (l'eixam), que anomenarem E . A cada partícula $p_i \in E$ se li assigna una **posició inicial** x_i en S segons la fórmula:

$$x_i \leftarrow U(\inf, \sup) \quad (60)$$

en què $U : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ és una funció que retorna un nombre aleatori amb distribució uniforme dins del subespai de \mathbb{R}^n delimitat pels dos punts que rep com a paràmetre.

A més cada partícula té una **velocitat inicial** v_i que està determinada per la fórmula:

$$v_i \leftarrow U(-|\sup - \inf|, |\sup - \inf|) \quad (61)$$

És a dir, la velocitat inicial en cada dimensió és proporcional a l'interval de l'espai de solucions en aquesta dimensió. Habitualment, la velocitat inicial es multiplica per un factor $0 < k < 1$ per a evitar que una velocitat inicial excessiva faci que les partícules se surtin de S en poques iteracions.

Finalment, cada partícula recorda el **millor punt** m_i que ha visitat; inicialment $m_i = x_i$, ja que és l'únic punt que ha visitat cada partícula. També és necessari emmagatzemar el millor punt visitat per l'eixam, g , és a dir, el **mínim global** obtingut fins un moment donat:

$$g \leftarrow \operatorname{argmin}_{m_i} f(m_i) \quad (62)$$

Iteracions de l'algorisme

Com és habitual en aquest tipus de mètodes, no hi ha un nombre predefinit d'iteracions; l'usuari del mètode haurà de triar. Les opcions més habituals són: nombre fix d'iteracions, repetir fins a assolir un valor de f suficientment bo, fins que no s'observin millors en g o fins que transcorri un temps determinat.

En cada iteració les partícules canvien la seva velocitat influïdes pel seu millor punt m_i i pel millor punt global en aquest moment, g . En primer lloc es calcula la nova velocitat de cada partícula de l'eixam $i = 1, \dots, |E|$ segons l'expressió:

$$v_i \leftarrow \alpha v_i + \beta_c a_c (m_i - x_i) + \beta_s a_s (g - x_i) \quad (63)$$

Els paràmetres i valors que apareixen en l'expressió anterior són:

- α és un coeficient que regula la **inèrcia** de la partícula, això és, la tendència a seguir amb la seva trajectòria i ignorar la influència d'altres elements.
- β_c (c de **cognitiva**) regula la influència de l'aspecte cognitiu de la partícula mateixa, és a dir, la influència del millor punt que recorda en el càlcul de la seva nova velocitat.
- β_s (s de **social**) regula la influència de l'aspecte social sobre la partícula, és a dir, la influència del millor punt trobat per l'eixam. Com més gran sigui, més tendiran les partícules a moure's cap al millor punt, g , i per tant la influència del social, de l'eixam, serà més gran.
- a_c i a_s són dos nombres aleatoris calculats mitjançant $U(0,1)$ que assignen una influència aleatòria als aspectes cognitius i socials, respectivament.

La nova posició de cada partícula depèn de la seva posició actual i de la seva velocitat nova:

$$x_i \leftarrow x_i + v_i \quad (64)$$

Si la nova posició de la partícula és una solució millor que la millor solució que recordava, és a dir, si $f(x_i) < f(m_i)$ llavors x_i passa a ser la nova posició millor d'aquesta partícula: $m_i \leftarrow x_i$.

Després d'actualitzar totes les partícules, se'n comparen les posicions millors m_i amb la millor posició global g ; si alguna $m_i < g$, llavors $g \leftarrow m_i$.

En acabar les iteracions, g conté la millor solució trobada per l'algorisme.

Selecció dels paràmetres α , β_c i β_s

En l'expressió 63 apareixen tres paràmetres que s'utilitzen per a calcular la nova velocitat d'una partícula. Aquests paràmetres permeten ajustar la influència de la inèrcia de la partícula (α), del millor punt que coneix (β_c) i del

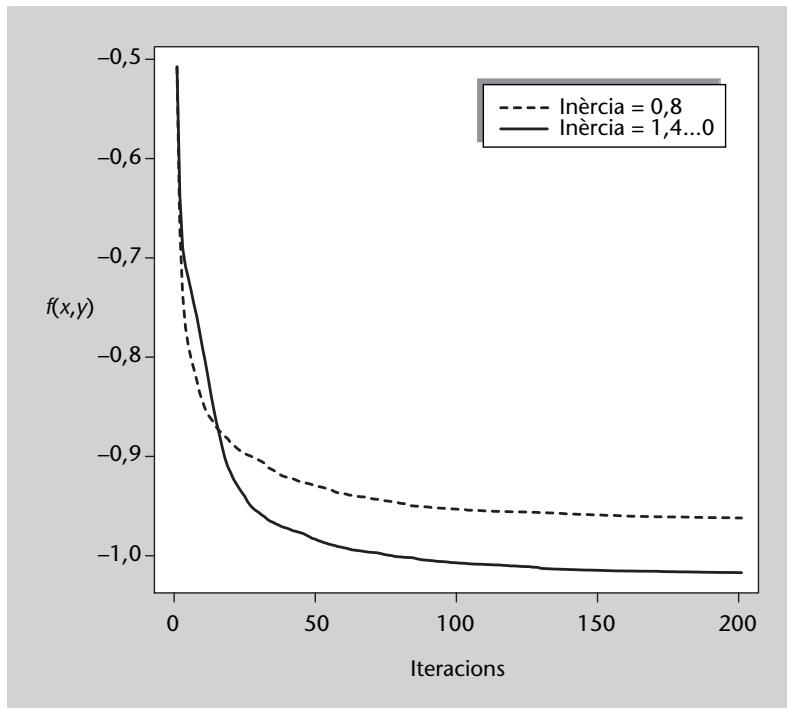
millor punt trobat per l'eixam (β_s). Ara bé, quin valor han de prendre? Diferents treballs empírics* recomanen que tant β_c com β_s tinguin un valor de 2,0, mentre que la inèrcia ha d'influir menys ($\alpha = 0,8$).

*Destaquem Y. Shi; R. C. Eberhart (1998). "Parameter selection in particle swarm optimization". Proceedings of Evolutionary Programming VII.

Una millora del mètode consisteix a començar amb una inèrcia relativament alta ($\alpha = 1,4$) que es va reduint en cada iteració, per exemple, multiplicant-la per un factor $r < 1$. El sentit d'aquesta millora és el següent. En les primeres iteracions les partícules es mouen a més velocitat a causa de l'elevada inèrcia; amb això s'aconsegueix explorar més exhaustivament l'espai de solucions S i evitar així quedar atrapat en mínims locals. No obstant això, a mesura que l'execució de l'algorisme avança, interessa polir les solucions trobades, duent a terme una exploració més fina del seu entorn. Per a aconseguir-ho les partícules s'han de moure lentament i explorar els voltants dels millors punts trobats, la qual cosa s'aconsegueix fent que la seva inèrcia sigui més petita.

Aquesta millora és bastant habitual i sol millorar els resultats dels algorismes d'optimització amb eixams de partícules. En la figura 59 es compara l'execució de l'algorisme bàsic (inèrcia constant) i de l'algorisme millorat (inèrcia decreixent). En aquest cas, es mostra el valor mitjà de $f(x,y)$ de 10 partícules durant 200 iteracions. S'ha triat un nombre relativament petit de partícules perquè l'obtenció de punts mínims no es degui a l'atzar inicial.

Figura 59. Comparació de les estratègies d'inèrcia constant i inèrcia decreixent



Com es pot observar, amb inèrcia constant l'algorisme comença millorant abans els resultats (minimitzant $f(x,y)$), si bé arriba un punt en el qual s'es-

tanca perquè no és capaç de fer una exploració més fina per a continuar millorant la posició de les partícules. Per la seva banda, amb inèrcia decreixent l'algorisme comença a optimitzar més lentament que amb inèrcia constant, però a mesura que l'execució avança continua millorant els resultats, ja que les partícules, ara més lentes, poden explorar amb més precisió l'àrea al voltant del mínim global. Per aquesta raó, el codi d'exemple del subapartat 5.6.4. fa ús d'inèrcia decreixent.

El valor del factor r depèn del nombre d'iteracions previstes, i convé ajustar-lo de manera que en les últimes iteracions la inèrcia tendeixi a zero, però no abans. Per exemple, amb 100 iteracions tenim que $0,9^{100} = 0,0000265$, i llavors $r = 0,9$ és adequat si el nombre d'iteracions és 100, però no si és, per exemple, 1000, ja que en aquest cas a partir de la iteració 100 la inèrcia queda pràcticament desestimada. En aquest cas seria millor prendre $r = 0,99$, per exemple, ja que en la iteració 100 la inèrcia continua tenint un pes raonable, $0,99^{100} = 0,366$.

5.6.2. Exemple d'aplicació

A manera d'exemple s'utilitzarà l'optimització amb eixams de partícules per a buscar el mínim de la funció 59. Es tracta d'un problema molt senzill, amb un espai de solucions molt "petit" (només dues dimensions i uns intervals bastant reduïts), per la qual cosa una simple exploració a l'atzar pot donar resultats acceptables. No obstant això, s'ha triat així expressament, ja que permet mostrar la traça d'execució completa, en aquest cas amb quatre partícules i tres iteracions.

En la taula 42 es pot veure, per a cada iteració, com les partícules es van movent cap a la solució i van millorant les seves posicions. Així, amb tan sols quatre partícules i tres iteracions, el mínim obtingut millora clarament, i passa de $-0,0372$ a $-0,927$, bastant proper al mínim global que ja coneixem.

5.6.3. Anàlisi del mètode

El mètode d'optimització amb eixams de partícules és un mètode adequat per a gran quantitat de problemes. Si bé l'exemple plantejat en el subapartat 5.6.2. permet comprovar pas per pas l'execució de l'algorisme en un cas senzill, la potència d'aquest mètode es fa patent en problemes amb més nombre de dimensions, en els quals la cerca a l'atzar resulta infructuosa. El mètode ofereix els avantatges següents:

- Nombre relativament reduït de paràmetres. En molts casos n'hi ha prou de triar un nombre de partícules, ja que el mètode sol funcionar bé assignant als paràmetres els valors $\alpha = 1,4 \dots 0$, $\beta_c = 2,0$ i $\beta_s = 2,0$, tal com s'ha explicat més amunt.

Taula 42. Execució de l'eixam de partícules

Partícula	Posició	Velocitat	$f(x,y)$	Millor posició	Millor $f(x,y)$
Eixam inicial					
1	(-1,463, 0,695)	(-0,236, -0,745)	0,195	(-1,463, 0,695)	0,195
2	(-0,0183, -0,101)	(-0,348, -0,211)	-0,0372	(-0,0183, -0,101)	-0,0372
3	(-1,625, -0,943)	(-0,164, -0,567)	3,197	(-1,625, -0,943)	3,197
4	(1,0491, -0,996)	(-0,555, -0,278)	1,225	(1,0491, -0,996)	1,225
g	(-0,0183, -0,101)		-0,0372		
Iteració 1					
1	(1,079, 0,182)	(2,542, -0,513)	2,405	(-1,463, 0,695)	0,195
2	(-0,297, -0,333)	(-0,279, -0,232)	-0,101	(-0,297, -0,333)	-0,101
3	(-0,400, -0,722)	(1,225, 0,222)	-0,123	(-0,400, -0,722)	-0,123
4	(-0,453, -1)	(-1,502, -0,754)	1,188	(-0,453, -1)	1,188
g	(-0,400, -0,722)		-0,123		
Iteració 2					
1	(0,641, -0,00817)	(-0,438, -0,190)	1,306	(-1,463, 0,695)	0,195
2	(-0,168, -0,379)	(0,129, -0,0454)	-0,317	(-0,168, -0,379)	-0,317
3	(0,580, -0,758)	(0,980, -0,0367)	-0,297	(0,580, -0,758)	-0,297
4	(-1,579, -1)	(-1,126, -0,830)	3,664	(-0,453, -1)	1,188
g	(-0,168, -0,379)		-0,317		
Iteració 3					
1	(-2, -0,932)	(-4,926, -0,924)	5,139	(-1,463, 0,695)	0,195
2	(-0,0640, -0,663)	(0,104, -0,285)	-0,927	(-0,0640, -0,663)	-0,927
3	(1,000847, -1)	(0,421, -0,824)	1,234	(0,580, -0,758)	-0,297
4	(-0,541, -0,0443)	(1,0381, 0,956)	1,0153	(-0,541, -0,0443)	1,0153
g	(-0,0640, -0,663)		-0,927		

Les coordenades expressen (x,y) . g és la millor posició global.

- Adequat per a espais de solucions continus.
- En haver-hi moltes partícules no es queda embussat en mínims locals.

D'altra banda, cal tenir en compte algunes limitacions o inconvenients del mètode:

- Pot no ser aplicable o resultar inefficient en espais de solucions discrets.
- Si el cost computacional de la funció objectiu és alt, aquest mètode incorrerà en elevats temps d'execució, ja que en haver-hi moltes partícules és necessari avaluar la funció objectiu moltes vegades.

Per il·lustrar el funcionament del mètode la figura 60 mostra el recorregut d'una partícula. Noteu com els primers canvis de posició són molt grans, i a mesura que se succeeixen les iteracions la partícula es mou amb salts cada vegada

més petits al voltant del punt mínim que està buscant. Aquest comportament està potenciat per l'ús d'inèrcia decreixent, si bé també ocorre –encara que en menys grau– quan s'usa inèrcia constant a causa de la proximitat més gran de la partícula al seu mínim local propi i al mínim global de l'eixam.

Figura 60. Recorregut d'una partícula

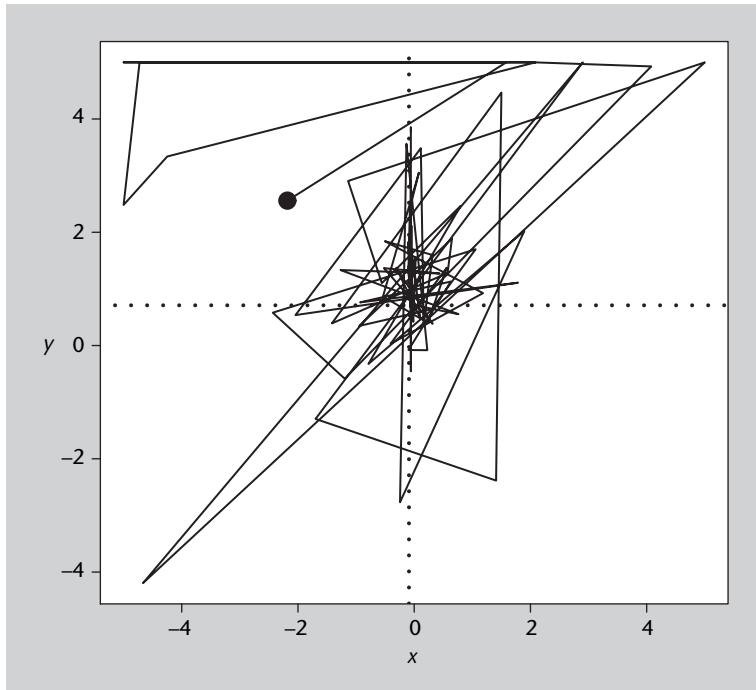


Figura 60

Recorregut d'una partícula des del punt inicial (assenyalat pel punt negre) cap al punt òptim (situat en la intersecció de les línies de punts). Es mostra el recorregut de la millor partícula d'una simulació amb 4 partícules i 100 iteracions.

5.6.4. Codi font en Python

Codi 5.4: Algorisme de l'eixam de partícules

```

1  from random import random
2
3  # Funció que es vol minimitzar
4  def funcion(x, y):
5      sum1 = x**2 * (4-2.1*x**2 + x**4/3.0)
6      sum2 = x*y
7      sum3 = y**2 * (-4+4*y**2)
8      return sum1 + sum2 + sum3
9
10
11 # Retorna un nombre aleatori dins d'un rang amb
12 # distribució uniforme (proporcionada per random)
13 def aleatori(inf, sup):
14     return random()*(sup-inf) + inf
15
16
17 # Classe que representa una partícula individual i que facilita
18 # les operacions necessàries
19 class Particula:
20     # Alguns atributs de classe (comuns a totes les partícules)
21     # Paràmetres per a actualitzar la velocitat
22     inercia      = 1.4
23     cognitiva    = 2.0
24     social       = 2.0

```

```

25 # Límits de l'espai de solucions
26 infx = -2.0
27 supx = 2.0
28 infy = -1.0
29 supy = 1.0
30 # Factor d'ajust de la velocitat inicial
31 ajustV = 100.0
32
33 # Crea una partícula dins dels límits indicats
34 def __init__(self):
35     self.x = aleatori(Particula.infx, Particula.supx)
36     self.y = aleatori(Particula.infy, Particula.supy)
37     self.vx = aleatori(Particula.infx/Particula.ajustV,
38                         Particula.supx/Particula.ajustV)
39     self.vy = aleatori(Particula.infy/Particula.ajustV,
40                         Particula.supy/Particula.ajustV)
41     self.xLoc = self.x
42     self.yLoc = self.y
43     self.valorLoc = funcio(self.x, self.y)
44
45 # Actualitza la velocitat de la partícula
46 def actualitzaVelocitat(self, xGlob, yGlob):
47     cogX = Particula.cognitiva*random()*(self.xLoc-self.x)
48     socX = Particula.social*random()*(xGlob-self.x)
49     self.vx = Particula.inercia*self.vx + cogX + socX
50     cogY = Particula.cognitiva*random()*(self.yLoc-self.y)
51     socY = Particula.social*random()*(yGlob-self.x)
52     self.vy = Particula.inercia*self.vy + cogY + socY
53
54 # Actualitza la posició de la partícula
55 def actualitzaPosicio(self):
56     self.x = self.x + self.vx
57     self.y = self.y + self.vy
58
59 # S'ha de mantenir dins de l'espai de solucions
60 self.x = max(self.x, Particula.infx)
61 self.x = min(self.x, Particula.supx)
62 self.y = max(self.y, Particula.infy)
63 self.y = min(self.y, Particula.supy)
64
65 # Si és inferior a la millor, l'adopta com a millor
66 valor = funcio(self.x, self.y)
67 if valor < self.valorLoc:
68     self.xLoc = self.x
69     self.yLoc = self.y
70     self.valorLoc = valor
71
72
73 # Mou un eixam de partícules durant les iteracions indicades.
74 # Retorna les coordenades i el valor del mínim obtingut.
75 def eixamParticules(particules, iteracions, reduccioInercia):
76
77     # Registra la millor posició global i el seu valor
78     millorParticula = min(particules, key=lambda p:p.valorLoc)
79     xGlob = millorParticula.xLoc
80     yGlob = millorParticula.yLoc
81     valorGlob = millorParticula.valorLoc
82
83     # Bucle principal de simulació
84     for iter in range(iteracions):
85         # Actualitza la velocitat i posició de cada partícula
86         for p in particules:
87             p.actualitzaVelocitat(xGlob, yGlob)
88             p.actualitzaPosicio()
89
90         # Fins que no s'han mogut totes les partícules no
91         # s'actualitza el mínim global, per a simular que totes es
92         # mouen alhora
93         millorParticula = min(particules, key=lambda p:p.valorLoc)
94         if millorParticula.valorLoc < valorGlob:
95             xGlob = millorParticula.xLoc
96             yGlob = millorParticula.yLoc

```

```

97     valorGlob = millorParticula.valorLoc
98
99     # Finalmente se reduce la inercia de las partículas
100    Particula.inercia*=reduccionInercia
101
102    return (xGlob, yGlob, valorGlob)
103
104
105 # Paràmetres del problema
106 nParticules = 10
107 iteracions = 100
108 redInercia = 0.9
109
110 # Genera un conjunt inicial de partícules
111 particules=[Particula() for i in range(nParticules)]
112
113 # Executa l'algoritme de l'eixam de partícules
114 print(eexamParticules(particules, iteracions, redInercia))

```

5.7. Cerca tabú

Una editorial està elaborant un atles del món i vol acolorir un mapa polític utilitzant un nombre determinat de colors i evitant que països contigus tinguin el mateix color, la qual cosa dificultaria la lectura del mapa. Com ha d'assignar un color a cada país?

El mètode de la **cerca tabú** és una millora de l'exploració aleatòria de l'espai de solucions S , l'avantatge del qual és que es recorden les solucions provades amb anterioritat. D'aquesta manera, no es tornen a explorar regions que ja s'han visitat prèviament, i així l'eficiència de cerca millora. El seu nom es deu al fet que les solucions ja analitzades es converteixen en tabú, ja que s'impedeix a l'algorisme accedir-hi.

5.7.1. Descripció del mètode

Suposem que volem buscar el mínim d'una funció $f : S \rightarrow \mathbb{R}$, en què S és l'espai de solucions possibles. Expressant-ho d'una altra manera, busquem el punt $x^* \in S$ que compleix $x^* = \operatorname{argmin} f(x) \forall x \in S$, o bé una aproximació acceptable.

Abans de descriure el mètode de la cerca tabú ens detindrem breument en el mètode de **cerca aleatòria local**, ja que totes dues estan estretament relacionades.

Cada punt $x \in S$ té un **veïnat** $V(x) \subset S$, que és el conjunt de punts que estan separats de x per un únic canvi. La definició de V depèn del problema concret que es tracti, però en general dos punts són veïns si només es diferencien en

Cerca aleatòria global

La cerca aleatòria global és diferent de la cerca aleatòria local, ja que consisteix a anar provant punts de S a l'atzar (sense cap relació entre ells) fins que es compleixi la condició de finalització.

una de les seves variables (possiblement en una quantitat fitada si es tracta de variables numèriques).

El mètode de la cerca aleatòria local comença avaluant un punt a l'atzar $x_0 \in S$. A continuació pren un punt veí a l'atzar $x_1 \in V(x_0)$, l'avalua i pren un veí de x_1 . El procés es repeteix fins que es compleix la condició de finalització que s'hagi establert.

És evident que el mètode de cerca aleatòria tornarà sovint sobre punts explorats prèviament, amb la pèrdua de temps consegüent. Aquí entra la millora de la cerca tabú: s'emmagatzemen els últims punts explorats, que queden marcats com a tabú, de manera que el mètode els evitarà i es veurà forçat a explorar àrees noves. Vist d'una altra manera, es defineix una nova funció de veïnat $V'(x,i)$ que depèn de la iteració i de l'algorisme, ja que exclou els veïns que s'han visitat amb anterioritat. En cada iteració es tria a l'atzar un punt $x_{i+1} \in V'(x_i,i)$, s'avalua i es marca com a tabú per a evitar visitar-lo en futures iteracions.

Atès que els espais de solucions soLEN tenir un nombre elevat de dimensions, la probabilitat que un punt torni a ser visitat és molt petita. Per aquesta raó, marcar punts de S com a tabú tindria un efecte pràcticament nul respecte a la cerca aleatòria local. Perquè la cerca tabú tingui un efecte pràctic les marques tabú no s'apliquen als punts complets, sinó a cadascun dels components dels punts. Així, si, per exemple, l'espai de solucions és \mathbb{Z}^3 , les marques tabú s'apliquen per separat als valors que prenen x , y i z . D'aquesta manera l'efecte tabú és més potent i la millora respecte a la cerca aleatòria local és apreciable.

Amb l'estrategia anterior es corre el risc de bloquejar molts punts, la qual cosa pot provocar que es perdin solucions valuoses. La manera d'evitar-ho és relaxar la qualitat de tabú dels valors, fent que les marques tabú durin només unes iteracions; seguidament, els valors tornen a estar disponibles. En el subapartat 5.7.3. s'estudia la influència de la durada de les marques tabú en l'eficiència del mètode.

Atès que es tracta d'un mètode que parteix d'un punt a l'atzar i explora l'espai de solucions mouent-se entre punts veïns, el mètode de cerca tabú es classifica com a mètode de cerca local.

5.7.2. Exemple d'aplicació

L'exemple plantejat inicialment, consistent a acolorir els països d'un mapa sense que dos països del mateix color es toquin, és un cas particular del pro-

blema més general de l'**acoloriment de grafs**. Com és habitual, el mapa que volem acolorir (figura 61) es pot veure com un graf en el qual cada país es representa mitjançant un node i els nodes corresponents a països fronterers estan units mitjançant un arc.

Figura 61. Problema de l'acoloriment d'un mapa

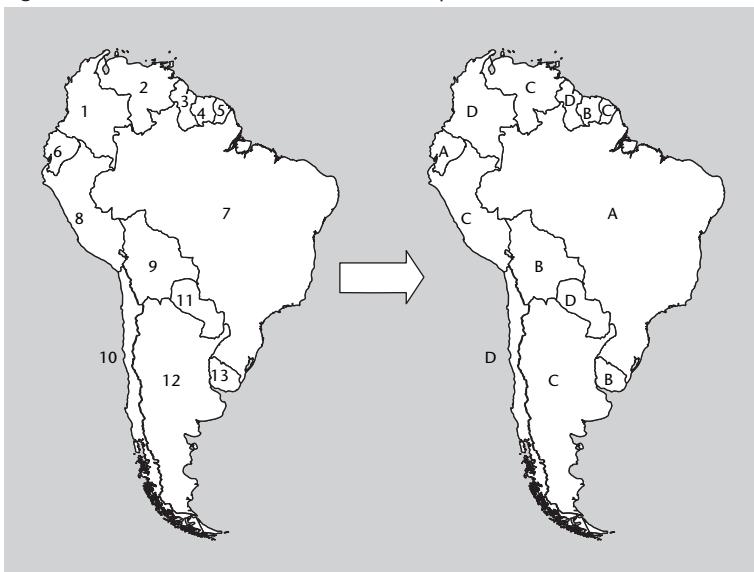


Figura 61

Exemple d'acoloriment d'un mapa: tenim un mapa amb tretze països (mapa esquerre) i el volem acolorir amb quatre colors (A, B, C i D). En el mapa dret es mostra un possible acoloriment del mapa obtingut mitjançant cerca tabú.

El problema genèric de l'acoloriment de grafs té moltes més aplicacions pràctiques, com l'assignació de registres del processador en compilat, o la gestió de recursos excloents com pistes d'aeroports, línies de producció, etc. En tractar-se d'un problema NP-complet, resulta interessant per a provar mètodes heurístics, atès que l'espai de solucions S té una mida $|S| = n^k$, en què n és el nombre de nodes i k el nombre de colors que es volen utilitzar; per aquest motiu, una exploració exhaustiva és impracticable excepte per a grafs molt petits.

Encara que la funció objectiu més evident per a resoldre aquest problema és el nombre de colors emprat en el mapa, aquesta funció no compleix les propietats enunciades en el subapartat 5.1.; en concret, el seu problema principal és que la seva resolució és molt grollera i, per tant, és incapàc de mesurar petites millores en l'acoloriment, amb la qual cosa els algorismes de cerca no reben cap realimentació positiva respecte a les millores que van trobant. És necessari utilitzar una funció objectiu de més resolució, i en aquest cas la candidata idònia és la funció que compta el nombre de col·lisions, això és, de països fronterers que tenen el mateix color. Com és evident, aquesta funció permet solucions no permeses, però a canvi proporciona una orientació adequada per a la cerca del mínim.

Les marques tabú s'apliquen a cadascun dels components de la solució. En aquest cas una solució és una correspondència d'un color per a cada país, per la qual cosa les marques tabú seran colors prohibits per a un país determinat, degudes al fet que recentment se li ha assignat aquest color.

Si volem obtenir el nombre mínim de colors amb els quals es pot acolorir un mapa donat, simplement hauríem d'iterar l'algorisme de cerca amb un nombre de colors decreixent, fins que l'algorisme sigui incapàc de trobar una assignació de colors vàlida (sense països fronterers del mateix color).

5.7.3. Anàlisi del mètode

La cerca tabú és un mètode conceptualment senzill i relativament eficient de millorar la cerca aleatòria local. No obstant això, es tracta d'un mètode de cerca local, amb les limitacions que això comporta; sovint convé executar-lo diverses vegades per a fer-lo més global i permetre-li trobar el mínim global.

Com tot mètode que utilitza una funció de veïnat, està més ben adaptat a espais de solucions discrets; en un altre cas, és necessari discretitzar l'espai o definir algun tipus de distància de veïnatge.

No hi ha un valor òptim per a la durada de les marques tabú, sinó que depèn del nombre de dimensions de l'espai de solucions i del nombre de valors possible de cada component. En la figura 62 s'estudia la influència de la durada de les marques tabú en l'eficiència del mètode en l'exemple de l'acoloriment del mapa. Una durada igual a zero equival a la cerca aleatòria local, que és clarament pitjor que la cerca tabú. Una durada molt curta (<10) provoca que el tabú tingui poc efecte. D'altra banda, una durada molt llarga torna el mètode molt rígid, ja que exclou nombroses solucions, i dificulta l'exploració. Per tant, tal com es veu en la figura, és millor triar un nombre intermedi d'iteracions, en aquest exemple concret 15.

Figura 62. Influència de la durada del tabú

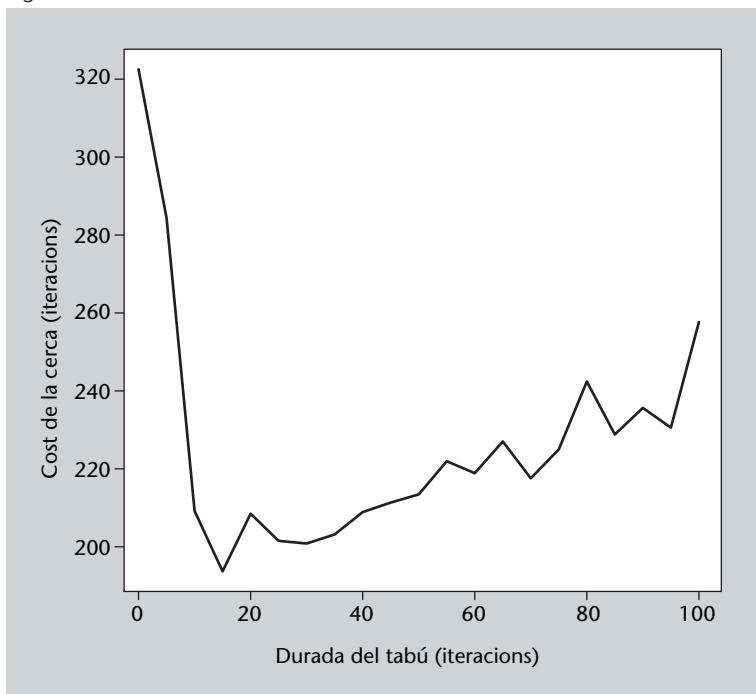


Figura 62

La durada del tabú influeix clarament en l'eficiència dels algorismes de cerca tabú. En la figura es mostra el nombre d'iteracions necessari per a obtenir la solució en un mapa amb 200 països, una probabilitat de contacte entre ells de 0,04 i cinc colors disponibles. Es mostren els valors mitjans de la cerca en 100 mapes diferents.

5.7.4. Codi font en Python

Codi 5.5: Acoloriment de mapes amb cerca tabú

```

1 # Nota: els grafs es representen com a llistes de llistes
2 from random import randint, random
3
4 # Dades dos nodes, retorna 1 si estan connectats, 0 si no
5 def conectats(graf, node1, node2):
6     if node1 < node2:
7         node1, node2 = node2, node1
8     if node1 == node2:
9         return 0
10    else:
11        return graf[node1][node2]
12
13
14 # Funció objectiu, que compta els vèrtexs connectats que tenen
15 # el mateix color
16 def objectiu(graf, estat):
17     suma = 0
18     for i in range(len(estat)-1):
19         for j in range(i+1, len(estat)):
20             if estat[i] == estat[j]:
21                 suma += connectats(graf, i, j)
22     return suma
23
24 # Retorna una llista amb els vèrtexs que tenen algun problema
25 # (estan connectats a un altre del mateix color)
26 def vertexsProblema(graf, estat):
27     nVertexs = len(estat)
28     vertexs = []
29     for i in range(nVertexs):
30         trobat = False
31         j = 0
32         while j < nVertexs and not trobat:
33             if (estat[i] == estat[j]) and connectats(graf, i, j):
34                 trobat = True
35             j += 1
36         if trobat:
37             vertexs.append(i)
38     return vertexs
39
40 # Genera la llista de veïns d'un estat i retorna el
41 # millor, tenint en compte la informació tabú i
42 # actualitzant-la per a anotar el canvi d'estat produït.
43 def vei(graf, estat, k, tabu, iterTabu):
44     nVertexs = len(estat)
45
46     # En primer lloc, s'obtenen tots els vèrtexs
47     # connectats a algun altre vèrtex del mateix color
48     vertexs = vertexsProblema(graf, estat)
49
50     # Per a cada un d'aquells parells, es proposa una
51     # nova solució consistent a canviar el color d'un
52     # dels vèrtexs (comprovant que no agafi un color tabú).
53     candidats = []
54     for v in vertexs:
55         # Si queden colors disponibles per a aquell vèrtex
56         if len(tabu[v]) < k:
57             nouColor = randint(1, k)
58             while tabu[v].has_key(nouColor):
59                 nouColor = randint(1, k)
60             nouEstat = estat[:]
61             nouEstat[v] = nouColor
62             candidats.append(nouEstat)
63
64     # Es retorna el millor estat obtingut (funció objectiu menor).
65     # Si no hi ha solucions possibles (els vèrtexs problemàtics
66     # tenen tots els colors en tabú) es retorna un canvi
67     # a l'atzar per a desbloquejar la situació.
68     if len(candidats) > 0:

```

```

69         escollit = min(candidats, key=lambda e:objectiu(graf,e))
70         posicio = candidats.index(escollit)
71         canviat = vertexs[posicio]
72     else:
73         escollit = estat [:]
74         canviat = sample(vertexs, 1)[0]
75         escollit[canviat] = randint(1, k)
76
77     # S'afegeix el canvi a la llista tabú.
78     tabu[canviat][estat[canviat]] = iterTabu
79     return escollit
80
81
82 # Actualitza la informació tabú, descomptant una iteració de
83 # totes les entrades i eliminant les que arriben a 0
84 def actualitzaTabu(tabu):
85     for tabuVertex in tabu:
86         for color in tabuVertex.keys():
87             tabuVertex[color] -= 1
88             if tabuVertex[color] <= 0:
89                 del tabuVertex[color]
90
91
92 # Donats un graf, una assignació inicial i un nombre de colors,
93 # busca una assignació òptima utilitzant cerca tabú.
94 # També cal indicar el nombre màxim d'iteracions, així
95 # com el nombre d'iteracions que es manté un color en tabú.
96 def cercaTabú(graf, assignacio, k, iteracions, iterTabu):
97
98     estat = assignacio [:]
99     millor = estat [:]
100    nVertices = len(estat)
101
102    # L'estructura "tabú" emmagatzema, per a cada vèrtex (posicions
103    # en la llista), els colors que té prohibits (claus de
104    # cada diccionari) i durant quantes iteracions ho estaran
105    # (valors de cada diccionari)
106    tabu = [] for n in range(nVertices)]
107    i = 0
108    while i < iteracions and objectiu(graf, estat) > 0:
109
110        # Selecciona el millor veí de l'estat actual
111        estat = vei(graf, estat, k, tabu, iterTabu)
112
113        if objectiu(graf, estat)<objectiu(graf, millor):
114            millor = estat [:]
115            print i,objectiu(graf, estat), objectiu(graf, millor)
116
117        # Actualitza els estats tabú (descompta una iteració)
118        actualitzaTabu(tabu)
119
120        # Avança a la iteració següent
121        i += 1
122
123    return millor
124
125
126 # Genera un graf en forma de matriu de connectivitat, amb un nombre
127 # de vèrtexs i una probabilitat de contacte entre dos vèrtexs.
128 # Retorna la meitat inferior de la matriu com a llista de llistes.
129 def generaGraf(nVertices, probContacte):
130     resultat = []
131     for i in range(nVertices):
132         # 1 si el nombre aleatori és menor o igual que
133         # la probabilitat de contacte, 0 si no
134         f = lambda : random()<=probContacte and 1 or 0
135         resultat.append([f() for j in range(i)])
136
137     return resultat
138
139
140 # Programa principal

```

```

141
142 # Exemple: graf que representa Amèrica del Sud (no es resol)
143 americadelsud = [[] ,
144     [1] ,
145     [0,1] ,
146     [0,0,1] ,
147     [0,0,0,1] ,
148     [1,0,0,0,0] ,
149     [1,1,1,1,1,0] ,
150     [1,0,0,0,0,1,1] ,
151     [0,0,0,0,0,1,1] ,
152     [0,0,0,0,0,0,1,1] ,
153     [0,0,0,0,0,0,1,0,1,0] ,
154     [0,0,0,0,0,0,1,0,1,1,1] ,
155     [0,0,0,0,0,0,1,0,0,0,1]]]
156
157 # Generació d'un graf aleatori.
158 # Paràmetres del problema: nombre de països i probabilitat
159 # que dos països qualssevol siguin fronterers.
160 # Lògicament, com més països hi hagi, més
161 # contactes tindrà un país determinat.
162 nPaisos      = 100
163 probContacte = 0.05
164 mapa         = generaGraf(nPaisos,probContacte)
165
166 # Execució de l'algoritme: nombre màxim d'iteracions
167 # de l'algoritme i nombre d'iteracions que roman un color
168 # com a tabú per a un país donat.
169 iteracions   = 200
170 iterTabu     = 15
171
172 # Nombre de colors
173 k=4
174
175 # Es crea un estat inicial aleatoriament
176 estat = [randint(1,k) for i in range(nPaisos)]
177
178 # S'executa la cerca tabú amb k colors.
179 solucio = cercatabu(prova, estat, k, iteracions, iterTabu)
180 print('Solucio:' + str(solucio))
181 print('Col·lisio=' + str(objectiu(prova, solucio)))

```

6. Annex: conceptes bàsics d'estadística

En l'apartat 3 vam definir la *densitat de probabilitat*. $\rho_x(k)$ en un interval b_k es defineix, llavors, com el nombre de valors que s'han produït en aquest interval h_k normalitzat amb l'àrea total de l'histograma

$$\rho_x(k) = \frac{h_k}{n\Delta x}, \quad (65)$$

i la *funció densitat de probabilitat contínua* prenent el límit en el qual la mida de l'interval Δx es fa infinitesimalment petit

$$\rho_x(x) = \lim_{\Delta x \rightarrow 0, m \rightarrow \infty} \rho_x(k). \quad (66)$$

Aquesta funció permet al seu torn definir la *funció de probabilitat acumulada*

$$p(x \leq x_0) = \int_{-\infty}^{x_0} \rho_x(x) dx, \quad (67)$$

que indica la probabilitat que x prengui un valor inferior o igual a x_0 . També permet definir els *moments estadístics* de la distribució de probabilitat de $x(t)$.

El *primer moment* o mitjana està determinat per

$$E[x] = \bar{x} = \int_{-\infty}^{\infty} x \rho_x(x) dx \quad (68)$$

que correspon al valor mitjà del senyal x , que representarem \bar{x} . Des d'un punt de vista geomètric, el valor mitjà \bar{x} pot ser interpretat com el centre de masses de la distribució de probabilitat ρ_x .

De manera equivalent, el *segon moment* de la distribució de $x(t)$ està determinat per

$$E[x^2] = \int_{-\infty}^{\infty} x^2 \rho_x(x) dx, \quad (69)$$

i pot ser expressat utilitzant la definició del primer moment com

$$E[x^2] = E[x]^2 + E[(x - E[x])^2] = \bar{x}^2 + E[(x - \bar{x})^2]. \quad (70)$$

El segon terme de l'última expressió es denomina *variància* de la distribució

$$\text{Var}[x] = E[(x - \bar{x})^2], \quad (71)$$

i la seva arrel quadrada és la *desviació típica*

$$\sigma = \sqrt{\text{Var}[x]}, \quad (72)$$

que constitueix una mesura de la variabilitat dels valors de $x(t)$ entorn del seu valor mitjà \bar{x} . En general, el moment d'ordre p de la distribució ρ_x estarà determinat per

$$E[x^p] = \int_{-\infty}^{\infty} x^p \rho_x(x) dx. \quad (73)$$

A part de la mitjana i la variància, dos dels moments més rellevants són la *asimetria*, definida com el moment central (moment de la variable $x - \bar{x}$) de tercer ordre

$$E[(x - \bar{x})^3] = \int_{-\infty}^{\infty} (x - \bar{x})^3 \rho_x(x) dx, \quad (74)$$

i la *curtosi*, definida com el quotient entre els moments de tercer i quart ordre amb una correcció perquè una distribució gaussiana tingui curtosi nul·la.

$$K = \frac{E[x^3]}{E[x^4]} - 3. \quad (75)$$

La curtosi permet quantificar la mesura en què una distribució de probabilitat es desvia d'un patró gaussià, i és un dels estadístics que s'utilitzen en el càlcul dels components independents del mètode ICA. El codi 6.1 explica com podem calcular els moments estadístics d'un conjunt de dades. En aquest exemple s'utilitzen dades gaussianes de mitjana $\mu = 10$ i desviació típica $\sigma = 5,5$ (la variància és $\sigma^2 = 30,25$). Al final del codi es mostren els valors mostrals estimats, que coincideixen amb els valors poblacionals corresponents.

Codi 6.1: Càlcul dels moments estadístics d'un conjunt de dades

```

1 from scipy import stats
2 import numpy as np
3
4 # dades gaussianes:
5 mu, sigma = 10, 5.5
6 y = mu + sigma*np.random.randn(10000)
7
8 # Estimació de moments:
9 # moment de primer ordre: mitjana
10 mitjana = y.mean()
11
```

```

12 # moment segon ordre: variància
13 var = y.var()
14
15 # moment tercer ordre: asimetria (skewness)
16 skew = stats.skew(y)
17
18 # moment quart ordre: curtosi
19 kurt = stats.kurtosis(y)
20
21 >>> print media, var, skew, kurt
22 10.0575611617 30.9623694469 0.0126290768091 0.025294666173

```

En cas d'haver de calcular els moments de la distribució a partir d'un conjunt discret de valors del senyal $\{x_1, x_2, \dots, x_n\}$, procedirem a fer una *estimació mostral*. Dos possibles estimadors mostrals (encara que no els únics) de la mitjana $\bar{x} = E[x]$ i la variància $\sigma = \sqrt{\text{Var}[x]}$ estan determinats, respectivament, per

$$\bar{x} \approx \frac{1}{n} \sum_{i=1}^n x_i \quad (76)$$

$$\sigma \approx \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2. \quad (77)$$

Els estimadors mostrals presenten un error respecte als valors reals de la distribució (valors poblacionals), ja que per a cada conjunt de valors $\{x_1, x_2, \dots, x_n\}$ s'obté un valor estimat diferent. L'error d'estimació consta típicament de dues parts, una deguda a la desviació en mitjana de l'estimador respecte al valor real, coneguda com a *biaix*, i una altra part que dóna compte de la dispersió dels valors estimats a partir de diferents conjunts de valors, coneguda com *variància*. Depenent de l'estimador mostral escollit, tindrem una certa ponderació d'error degut a biaix i degut a variància. Per exemple, és fàcil demostrar que mentre que $\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$ estima la desviació típica σ de manera esbiaixada, l'estimador

$$\sigma \approx \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2. \quad (78)$$

és un estimador no esbiaixat de la desviació típica d'una distribució.

Activitats

1. Visualitzeu un diagrama de dispersió de les puntuacions de pel·lícules d'usuaris amb correlacions molt baixes (propera a -1), nul·les i molt altes (properes a 1). Determineu aproximadament quina seria la línia d'ajust en cada cas.
2. Relacioneu les puntuacions de pel·lícules a Movielens amb l'edat: s'assemblen més les puntuacions d'edats properes que les puntuacions generals entre si?
3. Completeu l'aplicació de recomanació de pel·lícules. L'aplicació haurà de demanar opinió sobre 5 pel·lícules a l'usuari i a partir d'aquestes dades suggerir-li pel·lícules que li podrien agradar.
4. Preneu una imatge amb taques de punts (dibuixades amb qualsevol programa de dibuix) i obtingueu els grups corresponents utilitzant k -mitjanes i c -mitjanes difús.
5. Preneu una fotografia i intenteu delimitar zones similars utilitzant c -mitjanes difús.
6. Agrupeu les pel·lícules de la base de dades Movielens per les seves valoracions. Analitzeu si hi ha correlació amb el gènere de la pel·lícula (fitxer *u.genre*).
7. Utilizeu el nucli gaussià en el programa d'agrupament espectral en calcular la matriu de distància. Analitzeu la influència del paràmetre σ en el nombre de grups obtingut.
8. Substituïu la laplaciana no normalitzada per la laplaciana de recorregut aleatori (*random walk*) en el programa 2.8 i compareu els resultats obtinguts.
9. Modifiqueu la implementació (programa 4.1) del Naïve Bayes (subapartat 4.2.1.) aplicant la distribució gaussiana per a poder tractar atributs numèrics i apliqueu-ho al conjunt de dades de les flors (subapartat 4.3.1.).
10. Modifiqueu la implementació (programa 4.2) del kNN (subapartat 4.3.1.) perquè voti tenint en compte la ponderació d'exemples en funció de la seva proximitat a l'exemple de test.
11. Modifiqueu la implementació (programa 4.2) del kNN (subapartat 4.3.1.) perquè utilitzi la distància de Hamming i apliqueu-ho al conjunt de dades dels bolets (subapartat 4.2.1.).
12. Modifiqueu la implementació (programa 4.3) del classificador lineal basat en distàncies (subapartat 4.3.2.) perquè utilitzi la distància de Hamming i apliqueu-ho al conjunt de dades dels bolets (subapartat 4.2.1.).
13. Modifiqueu la implementació (programa 4.4) del classificador basat en *clustering* (subapartat 4.3.3.) perquè utilitzi la distància de Hamming i apliqueu-ho al conjunt de dades dels bolets (subapartat 4.2.1.).
14. Modifiqueu la implementació (programa 4.4) del classificador basat en *clustering* (subapartat 4.3.3.) canviant el mètode de *clustering* (vegeu el subapartat 2.3.).
15. Modifiqueu la implementació (programa 4.5) dels arbres de decisió (subapartat 4.4.1.) perquè inclogui el tractament dels punts de tall d'atributs numèrics i apliqueu-ho al conjunt de dades de les flors (subapartat 4.3.1.).
16. Implementeu l'algorisme ID3 (subapartat 4.4.1.) partint del programa 4.5 i apliqueu-lo al conjunt de dades de les flors (subapartat 4.3.1.).
17. Afegiu la funcionalitat de la poda de branques al programa fet en l'exercici anterior.
18. Dissenyeu les regles febles per a tractar el problema de les flors (subapartat 4.3.1.) i modifiqueu el programa 4.6 per a poder distingir la classe *iris-setosa* de les altres.
19. Afegiu la tècnica de l'*un contra tots** (descrit en el subapartat 4.5.4.) al programa 4.6 i apliqueu-lo al conjunt de dades de les flors (subapartat 4.3.1.).
20. Partint dels dos exercicis anteriors, mireu quin és millor sobre el conjunt de dades i si la diferència és estadísticament significativa.
21. Modifiqueu el programa 4.7 perquè calculi els dos algorismes del subapartat 4.5.1.
22. Afegiu el *kernel* polinòmic $((\|x - z\|^2 + 1)^d$ amb d equivalent al grau del polinomi) al programa 4.8.

* En anglès, *one vs all*.

23. Proceseu les dades del segon conjunt de categorització descrit en el subapartat 4.1.1. per a eliminar les paraules sense contingut semàntic utilitzant la llista de *stop words* de l'anglès descrita en el mateix subapartat i compareu el primer algorisme del subapartat 4.5.3. amb les dades sense processar.

24. Proceseu les dades del segon conjunt de categorització descrit en el subapartat 4.1.1. per a eliminar les paraules que no apareixen més de dues vegades en el corpus d'entrenament i compareu el primer algorisme del subapartat 4.5.3. amb les dades sense processar, amb les del problema anterior i utilitzant tots dos processos.

25. Feu els dos exercicis anteriors per al tercer conjunt de dades del subapartat 4.1.1. i apliqueu-ho amb el segon programa del subapartat 4.5.3.

26. Amplieu el programa 4.10 amb el *kernel* per a normalitzar descrit en el subapartat 4.5.3. i compareu-ho amb el programa sense normalitzar.

27. Proveu els diferents tipus de *kernels* de les *svm^{light}* modificant el programa 4.11.

28. Proveu els diferents tipus de *kernels* de les *libsvm* modificant el programa 4.12.

29. Amplieu el programa 4.13 amb el *kernel* per a la ponderació de termes amb la informació mútua (subapartat 3.1.3.).

30. Amplieu el programa 4.13 amb el *kernel* per a la ponderació de termes amb l'entropia (subapartat 4.4.1.).

31. Amplieu el programa 4.13 amb el *kernel* per a la ponderació de termes del *tf-idf* descrita en el subapartat 4.5.3.

32. Amplieu l'exercici anterior perquè inclogui el *kernel* per a la semblança entre termes i així completar el *tf-idf*.

33. Amplieu els programes 4.2 i 4.3 amb la validació creuada i compareu quin dels dos va millor per al conjunt de dades de les flors (subapartat 4.3.1.) i si la diferència és significativa utilitzant el test de Student.

34. Repetiu l'exercici anterior utilitzant el test del Wilcoxon en lloc del de Student.

35. Busqueu vuit conjunts de dades en el repositori de la UCI, afegiu-los als de les flors i bolets i utilitzeu el test de Friedman i la seva millora per a comparar els algorismes: Naïve Bayes, kNN, lineal, arbres de decisió, AdaBoost i SVM (codis 4.1, 4.2, 4.3, 4.5, 4.6 i 4.12).

36. Utilitzeu l'estadístic kappa de dos en dos sobre els algorismes: Naïve Bayes, kNN, lineal, arbres de decisió, AdaBoost i SVMs (codis 4.1, 4.2, 4.3, 4.5, 4.6 i 4.13) per a crear una matriu de valors sobre el tercer conjunt de dades del subapartat 4.1.1. i creeu un dendrograma per a comparar els diferents algorismes.

37. Resoleu cadascun dels problemes d'exemple dels mètodes metaheurístics amb els mètodes restants. Compareu els resultats obtinguts i el temps de computació requerit en cada cas.

38. Utilitzeu una imatge en escala de grisos a manera de mapa d'altituds, en què els píxels més clars representen punts més elevats en un mapa. Trobeu el camí entre dos punts del mapa que compleixi que el desnivell acumulat (total de pujada i baixada) sigui mínim utilitzant l'algorisme de la colònia de formigues (subapartat 5.5.). Noteu que abans d'aplicar l'algorisme cal transformar el mapa d'altituds en un graf.

39. Escriviu un programa que resolgui un sudoku utilitzant el mètode de la cerca tabú vist en el subapartat 5.7. Perquè la funció objectiu compleixi les propietats vistes en el subapartat 5.1. ha d'admetre sudokus amb errors i anar eliminant-los progressivament.

40. En un institut amb 4 grups d'alumnes i 5 professors cal trobar una distribució d'horaris compatible (cap professor no pot fer classe en dos grups alhora i cap grup no pot rebre dues classes alhora). Cada grup té 6 classes cada dia durant cinc dies a la setmana, que es divideixen en 10 assignatures, amb 3 hores setmanals cadascuna. En un mateix dia no poden tenir més d'una hora de la mateixa assignatura. Cada professor imparteix dues assignatures íntegrament, per la qual cosa fa un total de 24 hores de classe a la setmana. Cal obtenir un horari vàlid utilitzant algun dels mètodes vistos en aquest apartat.

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: <http://archive.ics.uci.edu/ml>). EUA: University of California, School of Information and Computer Science.

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

Segaran, T. (2007). *Programming Collective Intelligence*. EUA: O'Reilly.

Shawe-Taylor, J.; Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. RU: Cambridge University Press.

