UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
SECŢIA CALCULATOARE

VIZAT DECAN
Prof. Dr. Kalman Pusztai

VIZAT ŞEF CATEDRĂ
Prof. Dr. Ioan Alfred Leţia

METODE DE ÎNVĂŢARE FOLOSIND TEORIA CLUSTERIZĂRII

Absolvent: **Dobra Alin-Viorel**

1. CONŢINUTUL PROIECTULUI:
   (a) Piese scrise: Introducere, Elemente de logică fuzzy, Teoria clusterizării, Maşini fuzzy, Reţele neuronale, Maşini bazate pe clusteri, Implementare, Testare, Bibliografie.
   (b) Piese desenate:
   (c) Anexe: Listing program.
2. LOCUL DOCUMENTATIEI:
3. CONSULTANŢI: Horia Giurgiu
4. DATA EMITERII TEMEI: 1 noiembrie 1997
5. TERMEN DE PREDARE: 8 iunie 1998

CONDUCĂTOR PROIECT

SEMNĂTURĂ ABSOLVENT

Cuprins

Capitolul 1

Introducere

Această lucrare abordează problema aproximării funcţiilor neliniare dintr-o perspectivă uşor diferită faţă de metodele deja consacrate. Principalul scop urmărit este prezentarea din punct de vedere teoretic a ideilor şi în mai mică măsură aplicarea lor practică sau utilizarea lor efectivă.

Ne preocupă în primul r"and aproximarea funcţiilor în condiţiile în care singurele informaţii despre acestea sunt măsurători (perechi de date intrare-ieşire) eventual afectate de zgomot. Această problemă are o deosebit de mare importanţă în ştiinţele experimentale, fiind o metodă de sistematizare a informaţiilor conţinute în volume mari de date.

Abordăm această problemă din perspectiva inteligenţei artificiale, aproximarea funcţiilor putînd fi privită şi ca o metodă de reprezentare a cunoştinţelor. Punctul de plecare este pe de o parte *Teoria clusterizării* iar pe de altă parte abordări deja consacrate ale aceleiaşi probleme: *Reţele neuronale* şi *Maşinile fuzzy.*

Două tipuri de maşini sunt propuse: *maşina quasiconstantă* şi *maşina quasiliniară.* Noutatea nu constă at"at în forma acestor maşini sau în modul de utilizare a acestora (inferenţă) care sunt asemănătoare cu celelalte abordări mai sus menţionate, ci în modul în care se face *antrenarea* unor astfel de maşini utilizînd în exclusivitate *Teoria clusterizării.* Adăugăm astfel o nouă clasă de aplicaţii ale acestei teorii la cele (destul de puţine la număr) deja existente.

Abordarea noastră are certe avantaje, at"at din punct de vedere teoretic c"at mai ales practic, comparativ cu celelalte metode, dintre care amintim:

- modelele construite sunt transparente în sensul că reprezentarea este foarte uşor de *citit* de către un expert uman, spre deosebire de reţelele neuronale.

- procesul de învăţare este foarte rapid datorită utilizării *Teoriei clusterizării*, în comparaţie cu metoda gradientului
- problema alegerii dimensiunii optime a modelului este uşor de soluţionat, nefiind necesare luni de încercări ca în cazul reţelelor neuronale sau a maşinilor fuzzy
- nu este necesară alegerea unor parametrii de genul *rată de învăţare* care să diminueze din utilitatea algoritmilor sau să necesite zeci de încercări per problemă concretă pentru a determina o valoare convenabilă a lor
- nu necesită algoritmi sofisticaţi de oprire a învăţării pentru a evita pierderea generalizării datorită supraînvăţării, modelul comportîndu-se totdeauna bine şi pentru date pentru care sunt împrăştiate printre datele de antrenament (o interpolare de bună calitate)
- operaţiile matematice necesare at"at în procesul de învăţare c"at şi în cel de utilizare sunt cele elementare (adunare, scădere, înmulţire, împărţire) fiind posibilă cu efort minim o implementare hardware a algoritmilor. Nefiind necesară evaluarea unor funcţii neliniare derivabile (ca în cazul reţelelor neuronale) pe calculatoarele actuale pe care aceste operaţii sunt destul de costisitoare se va observa un spor important de viteză.

Toate acestea fac ca metoda propusă să fie un excelent subiect de studiu potrivit ca importanţă, complexitate şi noutate pentru o lucrare de diplomă. Întreaga abordare propusă este contribuţia personală a autorului; în afara articolului [7] ideile expuse aici nu au mai fost publicate nicăieri.

Lucrarea cuprinde 8 capitole grupate în trei părţi şi o anexă.

În prima parte se prezintă c"ateva aspecte matematice importante pentru demersul teoretic întreprins. Astfel capitolul 2 prezintă c"ateva elemente de logică fuzzy strict necesare urmate apoi în capitolul 3 de elemente de teoria clusterizării.

Partea a doua se ocupă cu problema aproximării funcţiilor prezentînd trei abordări ale inteligenţei artificiale pentru soluţionarea acesteia. Primele două, expuse în capitolele 4 şi 5 prezintă metode mult studiate în ultimul deceniu: reţelele neuronale şi maşinile fuzzy. Capitolul 6 este contribuţia autorului la acest domeniu şi se ocupă cu prezentarea teoretică a maşinilor quasiconstante şi quasiliniare.

Partea a treia este una mai practică şi are ca scop prezentarea programului `clus` şi a unor teste efectuate cu acesta. Capitolul 7 are ca scop expunerea pe larg a proiectării şi utilizării acestui program. O serie de experimente interesante care susţin ideile teoretice expuse pe tot cuprinsul lucrării sunt prezentate în capitolul 8.

Anexa conţine listingul programului `clus`, scris în întregime în C++.

At"at elaborarea programelor c"at şi tehnoredactarea acestei lucrări s-au făcut în întregime sub sistemul de operare *Linux* utilizînd programele gcc, LaTeX 2_ε şi XEmax.

Colaborarea cu domnul Conf. Dr. ing. Dorin Isoc la proiectul `MEDISIF` între anii 1993–1996 a făcut posibilă abordarea unei asemenea teme în cadrul lucrării de diplomă. Mulţumesc pe această cale domnului profesor căruia îi datorez iniţierea mea în inteligenţă artificială în general şi maşini fuzzy în special.

Mulţumesc de asemenea domnului Horia Giurgiu, conducătorul meu de diplomă, pentru sprijinul acordat în elaborarea acestei lucrări prin sfaturi şi observaţii interesante.

Discuţiile avute cu trei dintre colegii mei: Lucian Farcău, Mihai Hagianu şi Tiberiu Füstos, au dus la clarificarea unor aspecte importante legate de ideile expuse. Folosesc acest prilej să le mulţumesc şi lor de asemenea.

Partea I

Preliminarii matematice

În această secţiune a cărţii vom prezenta unele aspecte matematice fundamentale care să ne ajute în demersul nostru. Vom prezenta elemente de logică fuzzy şi de teoria clusterizării concentrîndu-ne asupra acelora care sunt strict necesare dezvoltărilor teoretice ce vor urma.

Deşi poate ar fi fost cazul să prezentăm şi c"ateva lucruri referitoare la geometria analitică şi algebră, lucruri la care se face referire pe cuprinsul lucrării, am presupus că acestea sunt suficient de cunoscute pentru cititorul avizat ca să fie suficiente referiri la cărţi care tratează mai pe larg aspectele atinse.

Fiind vorba de teorii consacrate din punct de vedere teoretic contribuţia autorului acestei lucrări este destul de mică şi se rezumă la o propunere de abordare într-o altă manieră (uşor diferită de cea consacrată) a inferenţei fuzzy şi la c"ateva metode de optimizare a algoritmilor de clusterizare, optimizări a căror contribuţie teoretică este mică dar care se dovedesc importanţi din punct de vedere practic.

Capitolul 2

Elemente de logică fuzzy

2.1 Introducere

Logica fuzzy se doreşte a fi o extindere naturală a logicii booleene (logica bivalentă cu două valori de adevăr: adevărat şi fals) în sensul introducerii unei infinităţi de valori de adevăr între fals (0) şi adevărat (1), valori de adevăr exprimate printr-un număr real din intervalul $[0,1]$. A fost fundamentată teoretic de către L. A. Zadeh (1965) şi este o extindere naturală a logicii trivalente şi mai apoi multivalente a lui Lukasiewicz (1934) şi a cercetărilor lui Menger (1951).

Necesitatea introducerii logicii fuzzy vine din faptul că în limbajul natural lucrurile nu sunt tranşante ca în cazul logicii booleene fiind astfel posibilă utilizarea cu succes a unor fapte, afirmaţii parţial adevărate sau imprecise. Logica fuzzy încearcă să surprindă şi să utilizeze o astfel de exprimare ambiguă a faptelor păstrînd însă rigurozitatea unui formalism matematic bine fundamentat.

O aplicaţie interesantă a logicii fuzzy este o teorie a mulţimilor fuzzy, teorie care extinde conceptele teoriei mulţimilor pentru mulţimi de obiecte care nu au graniţe nete. În loc să vorbind de apartenenţa sau neapartenenţa unui obiect la o mulţime vorbim de gradul de apartenenţă a acestuia. O astfel de extindere este şi ea conformă cu limbajul natural (ex: mulţimea oamenilor înalţi).

În această lucrare ne interesează în mod special noţiunea de mulţime fuzzy dar elemente generale de logică fuzzy ne sunt necesare pentru a elucida aspecte importante legate de maşinile fuzzy introduse în capitolul 4. Vom începe prin a prezenta succint teoria mulţimilor fuzzy definind operaţii pe aceste mulţimi (intersecţie şi reuniune) folosind formalismul normelor şi conormelor triunghiulare, continuînd apoi cu variabile lingvistice şi raţionament aproximativ.

2.2 Teoria mulţimilor fuzzy

Acest subcapitol prezintă c"ateva noţiuni fundamentale legate de teoria mulţimilor fuzzy.

Plecăm de la teoria mulţimilor clasice punînd-o într-o formă care permite generalizarea la mulţimi fuzzy.

Definim operaţiile pe mulţimi fuzzy folosind formalismul normelor şi conormelor triunghiulare încercînd să păstrăm proprietăţi ale mulţimilor clasice (legile lui De Morgan, comutativitate).

Introducem noţiunea de partiţie fuzzy ca rafinare a noţiunii de partiţie clasică.

2.2.1 Funcţia de apartenenţă pentru mulţimi clasice

Pentru a introduce operaţii pe mulţimi fuzzy punem mai înt"ai operaţiile pe mulţimi clasice într-o formă uşor de generalizat. Pentru detalii suplimentare se poate consulta [12] capitolul 9.

Ideea este de a exprima orice submulţime A a unei mulţimi X printr-o funcţie $f_A : X \longrightarrow \{0,1\}$ definită astfel:

$$f_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} \tag{2.1}$$

Dacă, aşa cum am sugerat în introducere, ataşăm valorii 1 interpretarea adevărat şi valorii 0 cea de fals, funcţia $f_A(x)$ exprimă de fapt validitatea afirmaţiei *Obiectul x aparţine submulţimii A*. Evident $f_X(x) = 1, \forall x \in X$ şi $f_\varnothing(x) = 0, \forall x \in X$.

Operaţiile pe mulţimi (intersecţia şi reuniunea ne interesează în mod deosebit) se pot translata în operaţii algebrice pe funcţii de forma celei prezentate mai sus. Astfel avem:

$$\begin{aligned} f_{A\cap B}(x) &= \min(f_A(x), f_B(x)) \\ &= f_A(x) f_B(x) \\ &= \max(f_A(x) + f_B(x) - 1, 0) \\ &= F(f_A(x), f_B(x)) \end{aligned} \tag{2.2}$$

şi

$$\begin{aligned} f_{A\cup B}(x) &= \max(f_A(x), f_B(x)) \\ &= f_A(x) + f_B(x) - f_A(x)f_B(x) \\ &= \min(f_A(x) + f_B(x), 1) \\ &= G(f_A(x), f_B(x)) \end{aligned} \tag{2.3}$$

unde $F : \{0,1\} \times \{0,1\} \longrightarrow \{0,1\}$ şi $G : \{0,1\} \times \{0,1\} \longrightarrow \{0,1\}$ sunt orice funcţii care au următoarele definiţii punctuale:

$$\begin{aligned} F(0,0) &= 0 \\ F(0,1) &= 0 \\ F(1,0) &= 0 \\ F(1,1) &= 1 \end{aligned} \tag{2.4}$$

şi

$$\begin{aligned} G(0,0) &= 0 \\ G(0,1) &= 0 \\ G(1,0) &= 0 \\ G(1,1) &= 1 \end{aligned} \tag{2.5}$$

şi pe deasupra au următoarele proprietăţi, proprietăţi derivate din proprietăţile operaţiilor pe mulţimi:

1. reuniunea şi intersecţia să fie comutative $\Longleftrightarrow$ funcţiile F şi G sunt comutative

2. $$\begin{matrix} A \cap X = A \\ A \cap \varnothing = \varnothing \end{matrix} \quad \Longleftrightarrow \quad \begin{matrix} F(a,1) = a \\ F(a,0) = 0 \end{matrix} \quad \forall a \in \{0,1\} \tag{2.6}$$

3. $$\begin{matrix} A \cup X = X \\ A \cup \varnothing = A \end{matrix} \quad \Longleftrightarrow \quad \begin{matrix} G(a,1) = 1 \\ G(a,0) = a \end{matrix} \quad \forall a \in \{0,1\} \tag{2.7}$$

4. Condiţiile De Morgan pentru mulţimi

$$\begin{matrix} \overline{A \cup B} = \bar{A} \cap \bar{B} \\ \overline{A \cap B} = \bar{A} \cup \bar{B} \end{matrix} \quad \Longleftrightarrow \quad \begin{matrix} F(a,b) = 1 - G(1-a, 1-b) \\ G(a,b) = 1 - F(1-a, 1-b) \end{matrix} \quad \forall a, b \in \{0,1\} \tag{2.8}$$

În relaţiile de mai sus am folosit complementara unei submulţimi faţă de o mulţime $\bar{A} = X/A$. Avem prin definiţie:

$$f_{\bar{A}}(x) = 1 - f_A(x) \tag{2.9}$$

O generalizare directă a formalismului de mai sus scris pentru mulţimi clasice se obţine lăsînd parametrii funcţiilor F şi G să aparţină intervalului $[0,1]$ în loc de a lua valori în mulţimea $\{0,1\}$.

2.2.2 Mulţimi fuzzy. Operaţii cu mulţimi fuzzy folosind funcţii de apartenenţă

Conform [12] o *mulţime fuzzy* este o pereche (X,A) unde X este *universul* mulţimii fuzzy iar $A : X \longrightarrow [0,1]$ este funcţia de apartenenţă care ataşează fiecărui element din X o valoare reală care reprezintă gradul de apartenenţă al elementului la mulţimea nuanţată (X,A).

În toate situaţiile din această lucrare universul discursului (universul mulţimilor fuzzy) este acelaşi pentru mulţimile cu care se fac operaţii (altfel operaţiile nu prea au sens) şi deci nu creăm nici o confuzie dacă identificăm mulţimea fuzzy cu funcţia ei de apartenenţă făcînd precizări suplimentare doar acolo unde acestea sunt necesare.

Următoarele proprietăţi ale mulţimilor fuzzy pot fi uşor obţinute prin generalizarea proprietăţilor mulţimilor clasice:

1. $A = B \Longleftrightarrow A(x) = B(x),\ \forall x \in X$
2. $A \subseteq B \Longleftrightarrow A(x) \leq B(x),\ \forall x \in X$
3. $\bar{A} = X/A \Longleftrightarrow \bar{A}(x) = 1 - A(x),\ \forall x \in X$
4. $\varnothing(x) = 0,\ \forall x \in X$
5. $X(x) = 1,\ \forall x \in X$

Facem aceiaşi generalizare şi în cazul funcţiilor F şi G din subcapitolul precedent. Proprietăţile 2.6 şi 2.7 păstrează aceiaşi formă cu diferenţa că variabilele a şi b care apar în acestea pot avea orice valoare în intervalul $[0,1]$. Aceiaşi generalizare o facem şi pentru ecuaţiile 2.8. Observăm totodată că aceste două ecuaţii se pot deduce una din alta deci ajunge să păstrăm una dintre ele ca şi condiţie impusă.

O condiţie suplimentară care trebuie impusă asupra funcţiilor F şi G c"and se face trecerea la mulţimi fuzzy este monotonia acestora în raport cu ambele variabile.

Cea mai largă clasă de funcţii care satisface condiţiile impuse pentru funcţiile F se numeşte clasa normelor triunghiulare sau a *t-normelor*. Pentru detalii vezi [12]. Funcţia G

care este obţinută din funcţia F folosind ecuaţia 2.8 se numeşte *t-conormă* duală. Pentru a conforma notaţia cu cea deja consacrată în literatura de specialitate vom nota cu T *t-norma* şi cu S *t-conorma*.

În literatura de specialitate au fost propuse o seamă de perechi *t-normă t-conormă*. Pentru o listă a celor mai utilizate vezi [15]. Dintre acestea o importanţă aparte o are perechea:

$$T(a,b) = \min(a,b) \tag{2.10}$$

$$S(a,b) = \max(a,b) \tag{2.11}$$

care are c"ateva proprietăţi remarcabile (vezi [12] pag. 239–240).

În funcţie de norme şi conorme reuniunea şi intersecţia se definesc în felul următor:

$$\begin{array}{l} (A \cap B)(x) = T(A(x), B(x)) \\ (A \cup B)(x) = S(A(x), B(x)) \end{array} \quad \forall x \in X \tag{2.12}$$

Făcînd notaţiile

$$T(x_1, ..., x_n, x_{n+1}) = T(T(x_1, ..., x_n), x_{n+1}) \tag{2.13}$$

$$S(x_1, ..., x_n, x_{n+1}) = S(S(x_1, ..., x_n), x_{n+1}) \tag{2.14}$$

şi ţinînd seama că intersecţia şi reuniunea mulţimilor este asociativă se poate arăta că

$$\bigcap_{i=1}^{n} A_i\,(x) = T(A_1(x), ..., A_n(x)) \tag{2.15}$$

$$\bigcup_{i=1}^{n} A_i\,(x) = S(A_1(x), ..., A_n(x)) \tag{2.16}$$

2.2.3 Partiţii fuzzy

Reamintim mai înt"ai definiţia noţiunii de partiţie din cadrul teoriei mulţimilor. Un set de mulţimi $A_1, ..., A_n$ formează o partiţie a mulţimii X dacă şi numai dacă următoarele condiţii sunt adevărate:

1. $A_1, ..., A_n$ sunt submulţimi ale mulţimii X

2. reuniunea mulţimilor $A_1, ..., A_n$ este mulţimea X

$$\bigcup_{i=1}^{n} A_i = X$$

3. intersecţia mulţimilor $A_1, ..., A_n$ două c"ate două este mulţimea vidă

$$\forall i, j \in \{1, 2, ..., n\},\ i \neq j,\ A_i \cap A_j = \varnothing$$

Dorim ca proprietăţile de mai sus să se păstreze şi în situaţia partiţiilor fuzzy. Se poate arăta (pentru detalii vezi [12] pag. 242–245) că setul $A_1, ..., A_n$ de mulţimi fuzzy formează o partiţie fuzzy a mulţimii fuzzy X, adică respectă proprietăţile enunţate mai sus, dacă şi numai dacă

$$\sum_{i=1}^{n} A_i(x) = 1,\ \forall x \in X \tag{2.17}$$

Această ecuaţie se va dovedi foarte importantă pentru *Teoria clusterizării fuzzy.* Se observă că ea are forma unei ecuaţii de normare din cadrul teoriei probabilităţilor.

2.2.4 Distanţa într-o mulţime fuzzy

Plecăm de la definiţia distanţei în spaţii metrice. Fiind dată o mulţime X, funcţia $d : X \times X \longrightarrow \mathbb{R}$ se numeşte distanţă sau metrică dacă următoarele condiţii sunt îndeplinite:

1. $d(x, x) = 0,\ \forall x \in X$
2. $d(x, y) = 0 \leftrightarrow x = y$
3. $d(x, y) = d(y, x),\ \forall x, y \in X$
4. $d(x, y) \leq d(x, z) + d(z, y),\ \forall x, y \in X$

Mulţimea X împreună cu distanţa d formează un spaţiu metric. Pentru detalii legate de spaţii metrice vezi [6]. Numim mulţimea X mulţime suport a spaţiului metric.

Am definit mai sus distanţa între două puncte. Dorim să definim o distanţă între două mulţimi A şi B (ambele submulţimi ale mulţimii suport X a unui spaţiu metric, (X, d)) în funcţie de distanţa în acel spaţiu metric. O posibilă definiţie este următoarea:

$$D(A, B) = \begin{cases} \inf\limits_{\substack{x \in A \\ y \in B}} d(x, y), & \text{dacă } A \neq \varnothing \text{ şi } B \neq \varnothing \\ 0, & \text{dacă } A = \varnothing \text{ şi } B = \varnothing \end{cases} \tag{2.18}$$

Pentru a defini distanţa dintre două mulţimi nuanţate trebuie să introducem noţiunea de *mulţime de nivel* t a lui A care este o mulţime clasică.

$$A^t = \{x \in X | A(x) \geq t\} \tag{2.19}$$

Acestea fiind spuse, distanţa dintre două mulţimi fuzzy A şi B definite pe acelaşi univers este

$$d(A,B) = \int_0^1 D(A^t, B^t)\, dt \tag{2.20}$$

Se poate arăta că definiţia de mai sus este validă în sensul că defineşte întradevăr o distanţă.

O altă noţiune utilă în cadrul logicii fuzzy este cea de *punct fuzzy*. Un punct fuzzy se defineşte ca fiind

$$f_y^a(x) = \begin{cases} a, & \text{dacă } x = y \\ 0, & \text{altfel} \end{cases} \tag{2.21}$$

Spunem că punctul fuzzy f_y^a aparţine mulţimii fuzzy A dacă şi numai dacă condiţia

$$f_y^a(x) \le A(x),\ \forall x \in X \tag{2.22}$$

este îndeplinită, ceea ce revine la $a \le A(y)$.

Se poate arăta uşor că distanţa dintre un punct fuzzy şi o mulţime fuzzy este

$$d(f_y^a, A) = \int_0^a D(y, A^T)\, dt \tag{2.23}$$

iar distanţa dintre două puncte fuzzy este

$$d(f_x^a, f_y^b) = \min(a,b)\ d(x,y) \tag{2.24}$$

2.2.5 Distanţa locală în raport cu o mulţime nuanţată

Din motive ce vor fi dezvăluite în capitolul 3 ne interesează să definim o distanţă locală în raport cu o mulţime nuanţată.

Fie (Y, d) un spaţiu metric şi $X \subseteq Y$. Fie A o mulţime fuzzy care are ca univers mulţimea X şi x şi y două puncte din X. Distanţa în raport cu mulţimea fuzzy A este funcţia $d_A : X \times X \longrightarrow \mathbb{R}$ definită ca

$$d_A(x,y) = d(f_x^{A(x)}, f_y^{A(y)}) \tag{2.25}$$

care ţinînd seama de ecuaţia 2.24 se poate rescrie:

$$d_A(x,y) = \min(A(x), A(y)) d(x,y) \tag{2.26}$$

Ne interesează să extindem această definiţie la tot spaţiul Y (distanţa pe o mulţime fuzzy induce o distanţă pe tot spaţiu). Pentru aceasta vom extinde mulţimea fuzzy A cu universul X la mulţimea A^* cu universul Y, legătura dintre A şi A^* fiind

$$A^*(x) = \begin{cases} A(x), & \text{dacă } x \in X \\ 1, & \text{altfel} \end{cases} \tag{2.27}$$

Fie $d_A^* : Y \times Y \longrightarrow \mathbb{R}$ extensia distanţei d_A la spaţiul Y care extinde definiţia 2.25 în modul următor:

$$d_A^*(x, y) = d(f_x^{A^*(x)}, f_y^{A^*(y)}),\ \forall x, y \in Y \tag{2.28}$$

care se poate rescrie astfel:

$$d_A^*(x, y) = \min(A^*(x), A^*(y))\ d(x, y) \tag{2.29}$$

Ecuaţia 2.29 poate lua diferite forme particulare. Astfel:

$$d_A^*(x, y) = \begin{cases} \min(A(x), A(y))\, d(x, y), & x, y \in X \\ A(x)\ d(x, y), & x, y \in Y, x \in X, y \notin X \\ A(y)\ d(x, y), & x, y \in Y, x \notin X, y \in X \\ d(x, y), & x, y \in Y, x, y \notin X \end{cases} \tag{2.30}$$

2.3 Raţionament aproximativ

În acest subcapitol vom aborda problema raţionamentului aproximativ sau fuzzy. În prima parte vom prezenta o perspectivă a acestui tip de raţionament acceptată de majoritatea comunităţii ştiinţifice. În partea a doua vom arăta neajunsurile, at"at teoretice c"at şi practice, ale unei astfel de abordări şi vom propune o alta care să remedieze unele din aceste neajunsuri.

Acest tip de raţionament este folosit de sistemele expert care î-şi propun să ţină seama de date imprecise provenite în special de la subiecţi umani.

2.3.1 Relaţii fuzzy

Am prezentat în subcapitolul anterior o teorie a mulţimilor fuzzy, teorie a cărei piatră de temelie era noţiunea de mulţime fuzzy. Se poate defini o altă noţiune utilă şi anume cea de *relaţie fuzzy*. O astfel de relaţie fuzzy exprimă intensitatea legăturii dintre perechi de elemente (x, y), unde $x \in X$ şi $y \in Y$, X şi Y fiind două mulţimi clasice.

Avem deci următoarea definiţie pentru relaţii fuzzy:

$$R : X \times Y \longrightarrow [0, 1] \tag{2.31}$$

unde $R(x, y)$ exprimă cât de puternică este legătura dintre x şi y.

Fie $* : [0, 1] \times [0, 1] \longrightarrow [0, 1]$ o operaţie cu următoarele proprietăţi:

1. simetrie
2. asociativitate
3. monotonie în ambele argumente

Se poate observa că toate *t-normele* şi t-conormele au aceste proprietăţi.

Putem defini acum compunerea relaţiilor $R : X \times Y \longrightarrow [0, 1]$ şi $S : Y \times Z \longrightarrow [0, 1]$ ca fiind:

$$(S \circ R)(x, z) = \sup_{y \in Y}(R(x, y) * S(y, z)) \tag{2.32}$$

O alegere uzuală pentru $*$ este min.

O altă operaţie interesantă este compunerea dintre o mulţime fuzzy şi o relaţie fuzzy, compunere care are ca rezultat o mulţime fuzzy. Fie $P : X \longleftarrow [0, 1]$ o mulţime fuzzy definită pe universul X şi $R : X \times Y \longrightarrow [0, 1]$ o relaţie fuzzy definită pe produsul cartezian $X \times Y$. Dacă $C = R \circ P$ atunci

$$C(y) = \sup_{x \in X}(P(x) * R(x, y)) \tag{2.33}$$

Această compunere se va dovedi utilă la deducerea unui mecanism de inferenţă pentru raţionamentul aproximativ.

2.3.2 Variabile lingvistice

O modalitate de a introduce raţionamentul aproximativ este prin intermediul variabilelor lingvistice. Variabilele lingvistice se doresc o punte de legătură între limbajul natural şi logica fuzzy.

O definiţie formală a noţiunii de *valabilă lingvistică* preluată din [12] este prezentată mai jos.

O *variabilă lingvistică* este un cvadruplu (T, G, U, M) unde

1. T e mulţimea valorilor lingvistice ale variabilei lingvistice v cu proprietatea că T e numărabilă
2. G e o gramatică care generează elementele din T. De obicei G e o gramatică independentă de context
3. U e universul discursului adică mulţimea suport a tuturor mulţimilor fuzzy ataşate valorilor lingvistice
4. M e regulă semantică ce ataşează unei valori lingvistice semnificaţia sa, sub forma unei mulţimi nuanţate peste universul U. $M : T \longrightarrow L(U)$ unde $L(U)$ e mulţimea mulţimilor fuzzy care au ca suport mulţimea U.

Exemple de variabile lingvistice sunt prezente în mai toate lucrările care încearcă să utilizeze logică fuzzy. Unele dintre cele mai interesante se pot găsi în [10],[2],[15].

Definiţia de mai sus este cea adoptată de majoritatea cercetătorilor, definiţie care lasă posibilitatea alegerii oricăror funcţii de apartenenţă neimpunînd nici o relaţie între ele.

Vom da mai jos o definiţie mai restrictivă dar care are avantaje care vor fi evidente din discuţiile care urmează.

Prin introducerea variabilelor lingvistice s-a încercat apropierea de modul de g"andire uman. Singurul lucru care, după părerea autorului, nu este apropiat de acest deziderat este noţiunea de funcţie de apartenenţă. Pentru a dezvolta puţin ideea dăm un exemplu: c"and spunem *om înalt* referindune la această noţiune ca la o valoare lingvistică nu ne g"andim că această noţiune are ataşată o funcţie de apartenenţă ci la faptul că există un obiect din universul discursului (multîmea suport) care este cel mai reprezentativ pentru mulţimea *oamenilor înalţi*. Pentru a putea spune *c"at de înalt* este omul x trebuie să generăm într-un fel său altul funcţia de apartenenţă echivalentă. Acest lucru se poate face considerînd o distanţă peste mulţimea suport şi luînd în considerare toate prototipurile ataşate valorii lingvistice în cauză. O astfel de abordare permite o interpretare mai naturală a noţiunii de valoare fuzzy. Valorile lingvistice pot fi considerate mulţimi fuzzy peste universul discursului.

O condiţie importantă ce se poate impune acestor mulţimii fuzzy este ca ele să formeze o partiţie a universului discursului. O astfel de condiţie este c"at se poate de naturală.

Rafinări de genul *foarte înalt* se transcriu în modificări (după o anumită reţetă) a prototipului. Spre deosebire de formalismul de mai sus în care întotdeauna c"and se face referire la o valoare lingvistică se poate utiliza doar funcţia sa de apartenenţă, în cazul formalismului propus de noi în situaţiile în care se vorbeşte de o variabilă lingvistică se utilizează prototipul acesteia, iar c"and variabila apare într-un predicat de genul *x este înalt* c"and ne interesează valoarea de adevăr a acestuia folosim funcţia de apartenenţă a valorii lingvistice. Această abordare aduce un surplus de coerentă şi o eficienţă mult sporită în procesul de raţionare aşa cum se va vedea în continuare.

2.3.3 Principiul Modus-Ponens generalizat

Ideea de bază de la care s-a plecat în realizarea raţionamentului aproximativ este generalizarea principiului Modus-Ponens din logica booleană, principiu care este o regulă de inferenţă completă pentru logica booleană. Exprimat în cuvinte acest principiu este: din faptul că propoziţiile A şi $A \longleftrightarrow B$ sunt adevărate se poate deduce că propoziţia B este adevărată.

Propoziţiile care ne interesează sunt de genul *x este A* unde A este o valoare lingvistică. În baza de cunoştinţe vom avea fapte de genul *Dacă x este Înalt atunci y este slab* şi av"and informaţii mai precise despre x ne interesează să deducem o valoare pentru y (numerică sau lingvistică).

Generalizarea care este acceptată de majoritatea cercetătorilor şi care foloseşte prima definiţie a variabilelor lingvistice este următoarea (vezi [10] pag.150–152): din (x este A^*) şi (dacă x este A atunci y este B) se poate deduce că (y este B^*), unde A,B,A^* şi B^* sunt valori lingvistice. Aceasta este de fapt o ecuaţie care are pe B^* ca şi necunoscută. Valoarea lingvistică A^* este echivalentul valorii numerice x şi se obţine printr-un procedeu numit *calificarea inversă a adevărului.* Din valoarea lingvistică B^* se poate deduce valoarea numerică y folosind *calificarea adevărului.*

Există mai multe metode de rezolvare a ecuaţiei pomenită mai sus. Prezentăm în continuare două dintre ele:

1. regulă compoziţională de inferenţă

$$B^*(y) = (R \circ A^*)(y) \tag{2.34}$$

$$= \sup_{x \in X}[\min(A^*(x), A(x), B(y))] \tag{2.35}$$

2. Inferenţă compoziţională folosind implicaţia Lukasiewicz

$$B^*(y) = (R \circ A^*)(y) \tag{2.36}$$

$$= \sup_{x \in X}[\min(1, A^*(x), 1 - A(x) + B(y))] \tag{2.37}$$

Aceste metode de inferenţă se obţin generalizînd valorile de adevăr (fie ele clasice sau fuzzy) la mulţimi fuzzy (valori lingvistice). O astfel de generalizare este oarecum forţată.

Folosind a doua definiţie (cea propusă de noi) a valorilor lingvistice putem generaliza principiul Modus-Ponens în felul următor: ştiind că *Dacă x este A atunci y este B* e o propoziţie adevărată şi *x este A* este adevărată într-un anumit grad atunci propoziţia *y este B* este adevărată în acelaşi grad. Cum fiecare valoare lingvistică are ataşat un prototip raţionamentul de mai sus mai poate fi formulat în felul următor: *y are o valoare cu at"at mai apropiată de P(B) cu c"at x are o valoare mai apropiată de P(A)* unde cu $P(A)$ şi $P(B)$ sa notat prototipul mulţimii A şi respectiv B.

Av"and mai multe reguli ca cea de sus în baza de cunoştinţe putem considera că valoarea lui y este o combinaţie convexă a prototipurilor concluziilor, ponderea fiecărui prototip fiind dictată de gradul de potrivire a concluziei. Astfel:

$$y = \sum_{i=1}^{r} w_i * y_i \tag{2.38}$$

unde w_i sunt ponderile prototipurilor y_i ale concluziilor, ponderi care se determină cu o formulă simplă din valorile $A_i(x)$. Am ţinut seama că avem r reguli de forma *Dacă x este A_i atunci y este B_i*, $y_i = P(B_i)$.

O astfel de inferenţă este mult mai eficientă din punct de vedere computaţional comparativ cu 2.34 sau 2.36 şi totodată este c"at se poate de naturală, trecerea de la Modus-Ponens la Modus-Ponens fuzzy fiind echivalentă cu trecerea de la mulţimi la mulţimii fuzzy.

Raţionamentul fuzzy va fi folosit în capitolele următoare pentru a prezenta aproximatori de funcţii.

Capitolul 3

Teoria clusterizării

În acest capitol vom prerezenta concepte generale de *Teoria Clusterizării* concepte necesare prezentării maşinilor bazate pe clusteri (vezi Cap. 6). Întreg acest capitol este inspirat din cartea domnului Dumitrescu ([9]). Ne vom mărginii doar la prezentarea aspectelor strict necesare, făcînd trimiteri directe la locul în care se pot găsi detalii. Fiind vorba de o teorie matematică bine fundamentată, contribuţiile teoretice ale autorului sunt limitate şi se referă doar la eficientizarea algoritmilor, fără a afecta aspectele fundamentale. Vom prezenta întotdeauna at"at varianta originală c"at şi cea optimizată acolo unde este cazul.

3.1 Noţiuni introductive

Teoria Clusterizării sau *Teoria clasificării* (cum mai este cunoscută) se ocupă în principal cu găsirea de reguli în volume mari de date. Posibilele aplicaţii ale unei asemenea teorii sunt din cele mai diverse, pornind de la *Inteligenţă artificială* şi continuînd cu *Psihologie*, *Biologie*, etc.

3.1.1 Problema fundamentală a Teoriei clusterizării

Dumitrescu sugerează în [9] pag. 5 următoarea formulare a problemei fundamentale a *Teoria Clusterizării* :

> D"andu-se o mulţime de obiecte $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^p\}$, aceste obiecte fiind caracterizate astfel înc"at este posibil să se definească o măsură a asemănării/neasemănării lor, să se găsească clasele de obiecte $A_1, A_2, ..., A_n$, $1 \leq n \leq p$ (de obicei aceste clase formează o partiţie a lui $\mathbf{X}$), astfel înc"at obiectele

aparţinînd aceleiaşi clase A_i să fie c"at mai apropiate posibile (similare), iar obiectele aparţinînd la clase diferite să fie c"at mai depărtate (nesimilare).

Nici un fel de restricţii legate de tipul datelor sau a măsurilor de similaritate/disimilaritate nu sunt necesare, posibilităţile metodei fiind deci foarte mari. Există posibilitatea comparării calităţii diferitelor clasificări folosind funcţii obiectiv. Calitatea clasificării se referă în primul r"and la omogenitatea în cadrul grupării şi separabilitatea între grupări.

Funcţiile obiectiv pot fi folosite în procesul de clasificare propriuzis nu doar în aprecierea calităţii clasificării, clasificarea putîndu-se face prin optimizarea unei funcţii criteriu. Acesta este de fapt singurul tip de clasificare (din acest punct de vedere) de care ne vom ocupa, fiind suficient de general şi convenabil de descris matematic. Exemple de funcţii obiectiv vor fi deci prezentate pe parcursul întregului capitol odată cu algoritmii de clusterizare.

Există mai multe posibilităţi de construire a funcţiilor criteriu. Putem considera de exemplu că fiecare clasă este reprezentată de un prototip geometric (un element tipic al clasei) şi să construim funcţia criteriu bazîndu-ne pe gradul de similaritate intre prototip şi elementele clasei pe de o parte şi gradul de disimilaritate între prototipul unei clase şi obiectele care nu aparţin clasei.

O altă metodă (de interes pentru lucrare ca şi cea descrisă mai sus) este *Analiza componentelor principale*. Ea constă în esenţă în luarea în considerare a extinderii spaţiale a claselor. Astfel se pot căuta în mulţimea datelor clase de formă alungită şi nu doar sferice ca în cazul metodei de mai sus.

Şi alte metode pentru construirea funcţiilor criteriu pot fi folosite dar acestea nu prezintă interes pentru scopul acestui capitol, acela de a pregăti *uneltele* necesare în capitolul 6.

3.1.2 Măsuri de disimilaritate

Pentru construcţia funcţiilor criteriu se pot folosi at"at măsuri de similaritate c"at şi de disimilaritate. Vom prezenta aici doar măsuri de disimilaritate, şi dintre acestea doar cele care ne interesează, exemple de alte măsuri de disimilaritate c"at şi de similaritate pot fi găsite în [9] pag. 10–12 şi 19–22.

Cea mai generală măsură de disimilaritate care poate fi definită peste o mulţime de date $\mathbf{X}$ este o pseudo-metrică (vezi [9] pag. 10). Suntem în primul r"and interesaţi de spaţii euclidiene (spaţii în care produsul vectorial este de forma $(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}$). De aceea vom considera pe parcursul lucrării că ne situăm doar în acest caz. Măsurile de disimilaritate pe care le vom introduce se bazează pe produsul vectorial, legătura dintre produsul vectorial şi norma spaţiului fiind:.

$$(\mathbf{x}, \mathbf{x}) = \|\mathbf{x}\|^2$$

C"at priveşte măsurile de disimilaritate ne interesează nu at"at disimilaritatea dintre două obiecte (puncte ale spaţiului) c"at disimilaritatea dintre un obiect şi un prototip de clasă (care nu întotdeauna este un obiect).

Astfel în cazul claselor cu de formă sferică, care au ca prototip un punct al spaţiului avem măsura de disimilaritate:

$$D(\mathbf{x}, \mathbf{L}) = \|\mathbf{x} - \mathbf{L}\|^2 \tag{3.1}$$

unde $\mathbf{x}$ este punctul din spaţiu corespunzător unui obiect din mulţimea obiectelor de clasificat iar $\mathbf{L}$ este prototipul unei clase din mulţimea claselor peste obiectele de clasificat. Se observă că disimilaritatea este cu at"at mai mare cu c"at obiectul este geometric mai depărtat de prototip.

Pentru clasele de formă liniară (şi deci cu prototipuri drepte) putem defini măsura de disimilaritate ca fiind distanţa de la un punct (obiectul) la o dreaptă (prototipul).

$$D(\mathbf{x}, \mathbf{L}) = \|\mathbf{x} - \mathbf{v}\|^2 - (\mathbf{x} - \mathbf{v}, \mathbf{u})^2 \tag{3.2}$$

Prototipul $\mathbf{L}$ este format dintr-un punct $\mathbf{v}$ şi o direcţie $\mathbf{u}$.

Un punct $\mathbf{v}$ şi r vectori independenţi $\mathbf{u}^1, ..., \mathbf{u}^r$ liniari independenţi într-un spaţiu euclidian formează o *varietate liniară* de ordinul r în acest spaţiu. Noţiunea de *varietate liniară* este o generalizare a noţiunii de dreaptă. Ecuaţia 3.2 se poate generaliza pentru varietăţi liniare astfel:

$$D(\mathbf{x}, \mathbf{L}) = \|\mathbf{x} - \mathbf{v}\|^2 - \sum_{k=1}^{r} (\mathbf{x} - \mathbf{v}, \mathbf{u}^k)^2 \tag{3.3}$$

Detalii şi demonstraţii ale formulelor de mai sus se pot găsi în [9].

3.1.3 Normalizarea datelor

După cum am arătat mai sus *Teoria Clusterizării* se foloseşte de funcţii criteriu care la r"andul lor folosesc norme (distanţe) euclidiene. Fiecărei dimensiuni a spaţiului îi corespunde un atribut măsurabil al obiectelor ce se doresc a fi clasificate. Unităţile de măsură şi domeniile în care aceste atribute pot lua valori pot să difere mult de la un atribut la altul, acest lucru ducînd la atribuirea unei importante mai mari unora dintre atribute în detrimentul celorlalte.

Pentru a evita o astfel de situaţie se impune normalizarea datelor, operaţie care constă în scalarea valorilor atributelor, astfel realizîndu-se o uniformizare a acestora. Normalizarea se face prin translaţii şi scalări transformînd fiecare coordonată a vectorului din spaţiul care are ca şi coordonate valori ale atributelor. Astfel:

$$x_i'^j = a_i x_i^j + b_i \tag{3.4}$$

unde a_i şi b_i sunt coeficienţii multiplicativi respectiv aditivi de normalizare pentru dimensiunea i, x_i^j este coordonata i a obiectului al j-lea, $x_i'^j$ este noua valoare a acestei coordonate, $1 \leq i \leq n$ cu n numărul de atribute şi deci şi de dimensiuni ale spaţiului, $1 \leq j \leq p$ cu p numărul de obiecte de clasificat.

Alegerea coeficienţilor de normalizare se poate face în mai multe moduri. Vom prezenta în continuare două dintre acestea.

Normarea la cubul $[0..1]^n$

În urma operaţiei de normare toate coordonatele vor avea valori cuprinse între 0 şi 1. Valoarea minimă pentru fiecare coordonată trece în 0 iar valoarea maximă în 1. Exprimînd algebric avem:

$$\begin{aligned} 0 =& a_i x^j_{i\,min} + b_i \\ 1 =& a_i x^j_{i\,max} + b_i \end{aligned} \tag{3.5}$$

de unde deducem:

$$\begin{aligned} a_i =& \frac{1}{x^j_{i\,max} - x^j_{i\,min}} \\ b_i =& \frac{-x^j_{i\,min}}{x^j_{i\,max} - x^j_{i\,min}} \end{aligned} \tag{3.6}$$

Această metodă de normalizare are dezavantajul că este "păcalită" de datele eronate, date care au valori cu mult depărtate de celelalte date, rezultatul normalizării putînd fi contrar celui scontat.

Normarea dispersaţională unitară

Această normare face ca datele, pe fiecare direcţie să aibă centrul de greutate (media) în 0 şi dispersia 1. Pătratul dispersiei este, prin definiţie media pătratelor abaterilor de la valoarea medie. În notaţie matematică aceasta se scrie:

$$\sigma^2 = \sigma^2(x_i) = \overline{(x_i - \bar{x}_i)^2} \tag{3.7}$$

Transformarea de normalizare a coordonatelor i a celui de al j-ilea obiect (i şi j au semnificaţia şi domeniul ca mai sus) este:

$$x_i'^j = \frac{x_i^j - \bar{x}_i^j}{\sigma_i} \tag{3.8}$$

Se poate arăta (vezi [9] pag. 18) că ecuaţia 3.8 este întradevăr o transformare de normalizare care are proprietatea că $\bar{x}_i'^j = 0$ şi $\sigma(x_i'^j) = 1$.

Putem exprima ecuaţia 3.8 în forma 3.4, coeficienţii a_i şi b_i fiind:

$$a_i = \frac{1}{\sigma_i} \tag{3.9}$$

$$b_i = \frac{\bar{x}_i^j}{\sigma_i} \tag{3.10}$$

3.2 Teoria clusterizării clasice

În continuare vom prezenta *Teorie a clusterizării clasice*, denumită astfel în contrast cu *Teoria clusterizării fuzzy* care va fi prezentată mai t"arziu în acest capitol. Acest subcapitol are ca scop introducerea c"atorva metode de împărţire a unei mulţimi de obiecte în clase disjuncte, clase care formează o partiţie în sensul teoriei mulţimilor a mulţimii de obiecte.

3.2.1 Funcţia criteriu

Spuneam mai devreme că metodele de clusterizare interesante din punct de vedere al acestei lucrări utilizează funcţii criteriu bazate pe măsuri de disimilaritate. Problema clasificării în contextul introducerii funcţiei criteriu se reformulează astfel: fiind dată o funcţie criteriu bazată pe o măsură de disimilaritate, se numeşte soluţie a problemei clasificării o partiţie a mulţimii de clasificat care face ca funcţia criteriu să aibă valoare minimă. Astfel problema clasificării se reduce la o problemă de minimizare a unei funcţii.

O noţiune fundamentală care serveşte la construirea funcţiei criteriu este noţiunea de prototip al unei clase. Prototipul unei mulţimi este reprezentantul acestuia (elementul tipic dar într-un sens mai larg). Notăm cu $\mathcal{L}$ mulţimea tuturor prototipurilor sau *spaţiul de reprezentare*. Fiind dată o măsură de disimilaritate D peste $\mathbf{X}$, mulţimea obiectelor de clasificat, se poate construi o disimilaritate între un obiect din $\mathbf{X}$ şi un prototip din $\mathcal{L}$ în condiţiile în care D este o distanţă sau pătratul unei distanţe (vezi [9]).

Fie D_i măsura de disimilaritate indusă de D, astfel $D_i : \mathbf{X} \times \mathcal{L} \longrightarrow \mathbb{R}$, $D_i(\mathbf{x}, \mathbf{L}_i)$ măsoară gradul de inadecvare între obiectul $\mathbf{x}$ şi prototipul $\mathbf{L}_i$. Mulţimea $\mathbf{L} = \{\mathbf{L}_1, \mathbf{L}_2, ..., \mathbf{L}_n\}$ este o reprezentare în spaţiul de reprezentare $\mathcal{L}$ a unei partiţii $\mathbf{P}$. Pentru situaţiile în care prototipul unei mulţimi este un punct din spaţiul caracteristicilor (ca şi fiecare element din $\mathbf{X}$ dealtfel) D şi D_i coincid.

Plecînd de la funcţia D_i care măsoară inadecvarea între un obiect şi o clasă (reprezentată prin prototipul său) putem construi o funcţie $I : P(\mathbf{X}) \times \mathcal{L} \longrightarrow \mathbb{R}$, funcţie ce măsoară gradul de inadecvare al reprezentării clasei A_i prin prototipul $\mathbf{L}_i$. Avem:

$$I(A_i, \mathbf{L}_i) = \sum_{\mathbf{x} \in A_i} D_i(\mathbf{x}, \mathbf{L}_i) \tag{3.11}$$

Putem introduce acum o funcţie care să sintetizeze gradul de inadecvare dintre o partiţie şi reprezentarea sa, $J : (P(\mathbf{X}))^n \times \mathcal{L}^n \longrightarrow \mathbb{R}$ de forma:

$$J(\mathbf{P}, \mathbf{L}) = \sum_{i=1}^{n} I(A_i, \mathbf{L}_i) \tag{3.12}$$

sau ţinînd cont de 3.11:

$$J(\mathbf{P}, \mathbf{L}) = \sum_{i=1}^{n} \sum_{\mathbf{x} \in A_i} D_i(\mathbf{x}, \mathbf{L}_i) \tag{3.13}$$

J este chear funcţia criteriu. Deoarece D_i este derivată dintr-o distanţă sau pătratul unei distanţe rezultă că $J(\mathbf{P}, \mathbf{L}) \geq 0$, ceea ce înseamnă ca această funcţie are un minim global pozitiv, fapt extrem de important ţinînd seama de faptul că se urmăreşte minimizarea acestei funcţii.

Clasificarea mulţimii de obiecte $\mathbf{X}$ se reduce la determinarea unei partiţii $\mathbf{P}$ cu reprezentarea $\mathbf{L}$ care minimizează această funcţie. Deşi problema poate fi rezolvată prin considerarea tuturor partiţiilor posibile peste o mulţime şi alegerea aceleia care are cea mai mică valoare pentru J, această metoda este neviabilă din punct de vedere calculaţional.

Găsirea minimului, chear şi local, printr-o metodă algebrică pentru o funcţie de forma funcţiei J este extrem de dificilă (dacă nu chiar imposibilă) şi nu este soluţionată în acest moment. Singurele metode viabile în acest caz sunt cele iterative. Ideea de bază a unei astfel de metode este următoarea: se porneşte de la o partiţie iniţială şi obiectele se mută dintr-o clasă în alta at"ata timp c"at mutarea ameliorează valoarea funcţiei criteriu. O astfel de metodă (din păcate) duce la găsirea unui minim local, pornirea algoritmului cu partiţii iniţiale diferite ducînd în general la rezultate finale diferite.

3.2.2 Metoda norilor dinamici

Folosind funcţii criteriu de această natură putem dezvolta un algoritm iterativ general de clusterizare care să nu ţină seama de forma explicită a măsurii de disimilaritate.

Ideea de bază a metodei norilor dinamici este minimizarea funcţiei J de forma 3.13 separat faţă de $P(\mathbf{X})$, partiţiile peste mulţimea de obiecte $\mathbf{X}$ şi $\mathbf{L}$ mulţimea de prototipuri, reprezentarea în spaţiul reprezentării a partiţiei $P(\mathbf{X})$. Minimizarea în acelaşi timp at"at faţă de $P(\mathbf{X})$ c"at şi faţă de $\mathbf{L}$ se dovedeşte a fi prea dificilă din punct de vedere algebric şi calculaţional.

În acest moment se pune întrebarea firească dacă metode de minimizare ca *metoda gradientului* nu sunt mai potrivite, ele fiind capabile s-a minimizeze funcţii de oric"ate variabile şi de orice formă. Funcţia J are o proprietate care poate fi din plin exploatată c"and se încearcă minimizarea ei: are o formă simplă (de multe ori pătratică) în $P(\mathbf{X})$ pe de o parte şi $\mathbf{L}$ pe de altă parte. Deci minimul lui J c"and $P(\mathbf{X})$ e constant se poate găsi analitic, la fel şi în situaţia în care $\mathbf{L}$ e constant.

Pentru a descrie procedeul iterativ de minimizare introducem două funcţii (vezi [9] pag. 25–26):

$$g : (P(\mathbf{X}))^n \longrightarrow \mathcal{L}^n \tag{3.14}$$

$$f : \mathcal{L}^n \longrightarrow (P(\mathbf{X}))^n \tag{3.15}$$

cu definiţiile:

$$g(\mathbf{P}) = \mathbf{L} = \{\mathbf{L}_1, \mathbf{L}_2, ..., \mathbf{L}_n\} \tag{3.16}$$

unde $\mathbf{L}_i$ realizează minimul funcţiei $I(A_i, ...)$, şi:

$$f(\mathbf{L}) = \mathbf{P} = \{A_1, A_2, ..., A_n\} \tag{3.17}$$

unde A_i minimizează funcţia $I(..., \mathbf{L}_i)$.

Funcţia g se numeşte *funcţie de reprezentare* (ataşează unei clase cea mai bună reprezentare) iar f se numeşte *funcţie de afectare* (face ca unei reprezentări să-i corespundă cea mai bună partiţie cu n clase).

Fie $\mathbf{L}^0$ o reprezentare iniţială arbitrară. Utilizînd funcţia f putem deduce o partiţie $\mathbf{P}^0 = f(\mathbf{L}^0)$. Din $\mathbf{P}^0$ prin intermediul funcţiei g putem găsi o reprezentare $\mathbf{L}^1 = g(\mathbf{P}^0)$, şi procesul continuă.

La pasul k avem:

$$\mathbf{L}^k = g(\mathbf{P}^{k-1}) \tag{3.18}$$

$$\mathbf{P}^k = f(\mathbf{L}^k) = f(g(\mathbf{P}^{k-1})) \tag{3.19}$$

Notăm:

$$a^k = (\mathbf{P}^k, \mathbf{L}^k) \tag{3.20}$$

$$b^k = J(a^k) \tag{3.21}$$

Şirurile a^k şi b^k descriu procesul iterativ de minimizare a funcţiei J. Un pas al acestui proces constituie o *iteraţie Picard*. Se poate arăta (vezi [9] pag. 27–29) că şirul b^k este descrescător şi convergent şi că în cazul *Teoriei clasice a clusterizării* şirurile a^k şi b^k devin staţionare după un număr finit de paşi.

3.2.3 Algoritmul n-medii

Vom particulariza acum metoda generală de clasificare prezentată în subcapitolul precedent pentru situaţia în care se caută clusteri sferici. Admitem că mulţimea $\mathbf{X}$ a obiectelor de clasificat reprezintă vectori d dimensionali dintr-un spaţiu euclidian (spaţiul valorilor atributelor). Ca măsură de disimilaritate considerăm:

$$D(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2 \tag{3.22}$$

În cazul clusterilor sferici, prototipul unei clase poate fi ales un punct din spaţiul euclidian considerat, av"and bineînţeles $D_i = D$ (măsura de disimilaritate indusă fiind chear măsura de disimilaritate considerată). Avem deci:

$$D(\mathbf{x}, \mathbf{L}_i) = \|\mathbf{x} - \mathbf{L}_i\|^2 \tag{3.23}$$

şi deci:

$$J(\mathbf{P}, \mathbf{L}) = \sum_{i=1}^{n} \sum_{\mathbf{x} \in A_i} \|\mathbf{x} - \mathbf{L}_i\|^2 \tag{3.24}$$

Notînd cu:

$$A_{ij} = \begin{cases} 1, & \text{dacă } \mathbf{x}^j \in A_{ij} \\ 0, & \text{altfel} \end{cases} \tag{3.25}$$

şi înlocuind în 3.24 şi ţinînd seama că produsul vectorial în spaţii euclidiene e de forma $(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}$ obţinem:

$$J(\mathbf{P}, \mathbf{L}) = \sum_{i=1}^{n} \sum_{j=1}^{p} A_{ij} (\mathbf{x}^j - \mathbf{L}_i)^\top M((x)^j - \mathbf{L}_i) \tag{3.26}$$

Punînd condiţia de minim faţă de $\mathbf{L}_i$ deducem:

$$\mathbf{L}_i = \frac{\sum_{j=1}^{p} A_{ij} \mathbf{x}^j}{\sum_{j=1}^{p} A_{ij}}, \ i = 1..n \tag{3.27}$$

Notăm cu:

$$p_i = \sum_{j=1}^{p} A_{ij} \tag{3.28}$$

numărul de elemente din clasa A_i. Din 3.27 şi 3.28 deducem:

$$\mathbf{L}_i = \frac{1}{p_i} \sum_{\mathbf{x} \in A_i} \mathbf{x} \tag{3.29}$$

Se observă că $\mathbf{L}_i$ este centrul de greutate al mulţimii A_i.

Din reprezentarea $\mathbf{L}_i$ se poate deduce o nouă mulţime A_i folosind metoda celui mai apropiat vecin. Acestea fiind spuse procedeul complet de determinare a clusterilor este descris în Algoritmul 1.

Observaţie: În loc să plecăm de la o partiţie iniţială se poate pleca de la o reprezentare iniţială care în acest caz poate fi o alegere aleatoare a n puncte din $\mathbf{X}$.

O astfel de procedură de clusterizare este potrivită doar pentru date care formează grupări aproximativ sferice, numărul de grupări fiind chear valoarea lui n din algoritm. Cum anume se determină numărul de grupări (aşazisa *problemă a validităţii clusterilor* nu face subiectul acestui algoritmi. Pentru o discuţie mai detaliată legată de problemele care mai pot interveni vezi [9] pag. 33-34.

3.2.4 Analiza componentelor principale

Acest subcapitol pune la dispoziţia unei noi metode de clasificare (algoritmul de clasificare cu prototipuri varietăţi liniare) instrumentele necesare. Dorim să luăm în considerare forma clusterilor în procesul de clasificare, în speţă să considerăm nori care nu sunt sferici

Algoritmul 1 Algoritmul n-medii

Require: $\mathbf{P}^0 = \{A_1, A_2, ..., A_n\}$ o partiţie iniţială a lui $\mathbf{X}$

Ensure: O partiţie care minimizează funcţia J

repeat

  for all mulţimile A_i ce formează partiţia $\mathbf{P}$ **do**

  $$\mathbf{L}_i = \frac{\sum_{j=1}^{p} A_{ij}\mathbf{x}^j}{\sum_{j=1}^{p} A_{ij}} = \tfrac{1}{p_i}\sum_{\mathbf{x}\in A_i}\mathbf{x} \ \{\text{prototipul mulţimii } A_i\}$$

  $$A_i = \begin{cases} 1, & \text{dacă } \left\|\mathbf{x}^j - \mathbf{L}_i\right\| < \left\|\mathbf{x}^j - \mathbf{L}_k\right\|, \forall k \neq i \\ 0, & \text{altfel} \end{cases}$$

  {calculul noii partiţii}

  end for

until partiţia calculată nu diferă de precedenta

ci pot avea alungiri mai pronunţate în anumite direcţii. *Analiza componentelor principale* se ocupă chear cu determinarea acestor direcţii.

Fiind dată o mulţime de puncte $\mathbf{X} = \{\mathbf{x}^1, ..., \mathbf{x}^p\}$ formînd un nor se doreşte să se determine direcţiile $\mathbf{u}^1, \mathbf{u}^2, ...$ de dispersie maximă a norului $\mathbf{X}$.

Direcţiile $\mathbf{u}^1, \mathbf{u}^2, ...$ sunt direcţii principale ale norului $\mathbf{X}$ dacă punctele acestuia aderă str"ans la dreptele care trec prin centrul de greutate al norului şi au direcţiile $\mathbf{u}^1, \mathbf{u}^2, ...$

Se ştie din geometria analitică că o varietate liniară de ordin m (generalizare a dreptei şi planului) poate fi descrisă printr-un punct şi m direcţii. Dorim determinarea unei varietăţi liniare astfel înc"at suma pătratelor distanţelor punctelor mulţimii X la ea să fie minimă (un criteriu de genul cele mai mici pătrate):

$$J = \sum_{j=1}^{p} d_j^2 \tag{3.30}$$

Se poate arăta (vezi [9] pag.64-66) că centrul de greutate al norului aparţine varietăţii liniare şi că direcţiile varietăţii sunt vectorii proprii corespunzători celor mai mari m valori proprii ai matricii:

$$S = \sum_{j=1}^{p} (\mathbf{x}^j - \mathbf{m})(\mathbf{x}^j - \mathbf{m})^\top \tag{3.31}$$

Algoritmul 2 Determinarea componentelor principale

Require: $\mathbf{X} = \{\mathbf{x}^1, ..., \mathbf{x}^p\}$ o mulţime de puncte

Ensure: $\mathbf{m}$ şi $\mathbf{u}^1, \mathbf{u}^2, ..., \mathbf{u}^m$ un punct şi m direcţii care definesc varietatea liniară care minimizează funcţia J definită cu formula 3.30

1: Se standardizează datele efectuînd transformarea $\mathbf{x}'^j = \mathbf{x}^j - \mathbf{m}$

$$\mathbf{m} = \frac{1}{p}\sum_{j=1}^{p}\mathbf{x}^j$$

2: Se determină vectorii proprii $\mathbf{u}^1, \mathbf{u}^2, ..., \mathbf{u}^p$ ai matricii de împrăştiere

$$S = \sum_{j=1}^{p}\mathbf{x}'^j\mathbf{x}'^{j\top}$$

3: Varietatea liniară de cea mai bună aproximare a norului X este determinată de punctul $\mathbf{m}$ şi direcţiile $\mathbf{u}^1, \mathbf{u}^2, ..., \mathbf{u}^m$ corespunzătoare celor mai mari valori proprii.

unde $\mathbf{m}$ este centrul de greutate al norului:

$$\mathbf{m} = \frac{1}{p}\sum_{j=1}^{p}\mathbf{x}^j \tag{3.32}$$

Algoritmul de determinare a componentelor principale este prezentat în Algoritmul 2. Se mai impune o observaţie. Valorile proprii ale matricii S pot indică de fapt c"at de împrăştiate sunt datele într-o anumită direcţie. Acest lucru nu este esenţial în cadrul acestei metode dar se va dovedi folositor mai t"arziu.

3.2.5 Clasificare cu prototipuri varietăţi liniare

Pentru a dezvolta o metoda de clasificare cu prototipuri varietăţi liniare urmăm modelul general prezentat în subcapitolul 3.2.2.

Prototipurile claselor (notate $\mathbf{L}$ ca şi p"ană acum) sunt în acest caz varietăţi liniare descrise de un punct $\mathbf{v}$ şi m direcţii, $\mathbf{u}^1, ..., \mathbf{u}^m$. Măsura de disimilaritate dintre un punct şi prototipul unei clase o alegem ca fiind pătratul distanţei de la punct la varietatea liniară:

$$D(\mathbf{x}, \mathbf{L}_i) = \left\|\mathbf{x}, \mathbf{v}^i\right\|^2 - \sum_{k=1}^{m}(\mathbf{x} - \mathbf{v}^i, \mathbf{u}^{ki})^2 \tag{3.33}$$

Folosind notaţia 3.25 şi expresia generală a funcţiei criteriu J, 3.12, putem scrie:

$$J(\mathbf{P},\mathbf{L}) = \sum_{i=1}^{n}\sum_{j=1}^{p} A_{ij}\left[\left\|\mathbf{x}^j,\mathbf{v}^i\right\|^2 - \sum_{k=1}^{m}(\mathbf{x}^j-\mathbf{v}^i,\mathbf{u}^{ki})^2\right] \tag{3.34}$$

sau ţinînd seama de expresia produsului scalar în spaţii euclidiene:

$$J(\mathbf{P},\mathbf{L}) = \sum_{i=1}^{n}\sum_{j=1}^{p} A_{ij}\left[(\mathbf{x}^j-\mathbf{v}^i)^\top(\mathbf{x}^j-\mathbf{v}^i) - \sum_{k=1}^{m}((\mathbf{x}^j-\mathbf{v}^i)^\top\mathbf{u}^{ki})^2\right] \tag{3.35}$$

Algoritmul 3 Clasificare folosind varietăţi liniare

Require: $\mathbf{P}^0 = \{A_1, A_2, ..., A_n\}$ o partiţie iniţială a lui $\mathbf{X}$

Ensure: O partiţie care minimizează funcţia J

1: **repeat**

2: **for all** mulţimile A_i ce formează partiţia $\mathbf{P}$ **do**

3: Se calculează reprezentarea $\mathbf{L}_i = (\mathbf{v}^i, \{\mathbf{u}^{1i}, ..., \mathbf{u}^{mi}\})$ pentru clasa A_i

$$\mathbf{v}^i = \frac{1}{p_i}\sum_{j=1}^{p} A_{ij}\mathbf{x} = \mathbf{m}^i$$

$\mathbf{u}^{1i}, ..., \mathbf{u}^{mi}$ vectori proprii corespunzători celor mai mari m valori proprii ai matricii

$$S_i = \sum_{j=1}^{p} A_{ij}(\mathbf{x}^j - \mathbf{m}_i)(\mathbf{x}^j - \mathbf{m}_i)^\top$$

4: Se calculează noua partiţie

$$A_{ij} = \begin{cases} 1, & \text{dacă } \mathbf{L}_i \text{ e cel mai apropiat prototip din } \mathbf{L} \\ 0, & \text{altfel} \end{cases}$$

5: **end for**

6: **until** partiţia calculată nu diferă de precedenta

Minimizarea funcţiei J se face ca şi în cazul general printr-o metodă iterativă constînd într-o secvenţă de paşi Picard, minimizînd funcţia pe r"and faţă de $\mathbf{P}$ şi faţă de $\mathbf{L}$.

Se poate arăta (vezi [9] pag. 75–79) că $\mathbf{L}_i = (\mathbf{v}^i, \{\mathbf{u}^{1i}, ..., \mathbf{u}^{mi}\})$ este un minim local al funcţiei $J(\mathbf{P}, .)$ dacă şi numai dacă:

$$\mathbf{v}^i = \frac{1}{p_i}\sum_{\mathbf{x}\in A_i}\mathbf{x} = \frac{1}{p_i}\sum_{j=1}^{p} A_{ij}\mathbf{x} = \mathbf{m}^i \tag{3.36}$$

şi $\mathbf{u}^{1i}, ..., \mathbf{u}^{mi}$ sunt vectorii proprii corespunzători celor mai mari m valori proprii ale matricii de împrăştiere definită cu formula:

$$S_i = \sum_{j=1}^{p} A_{ij} (\mathbf{x}^j - \mathbf{m}_i)(\mathbf{x}^j - \mathbf{m}_i)^\top \tag{3.37}$$

Determinarea partiţiei $\mathbf{P}$ c"and reprezentarea $\mathbf{L}$ este fixată (minimizarea funcţiei $J(., \mathbf{L})$) se face cu formula:

$$A_{ij} = \begin{cases} 1, & \text{dacă } \mathbf{L}_i \text{ e cel mai apropiat prototip din } \mathbf{L} \\ 0, & \text{altfel} \end{cases} \tag{3.38}$$

Algoritmul de clasificare cu prototipuri varietăţi liniare e prezentat schematic în Algoritmul 3.

3.3 Clasificare cu partiţii nuanţate

În majoritatea cazurilor practice mulţimea de puncte nu formează clase distincte, partiţie în sensul teoriei mulţimilor, dar sunt prezente aglomerări a căror graniţă nu poate fi delimitată cu precizie.

Pe de altă parte în cazul teoriei clasice a clusterizării toate punctele dintr-o clasă influenţează în aceiaşi măsură prototipul clasei chear dacă se află foarte aproape de graniţa cu alte clase, fiind deci mai puţin importante în comparaţie cu punctele apropiate de centrul norului.

Un alt aspect de care trebuie să se ţină seama este faptul că într-o seamă de aplicaţii practice prototipurile claselor sunt mai importante dec"at clasele ca atare.

Aceste lucruri fac ca generalizarea teoriei clasice a clusterizării într-o teorie fuzzy a clusterizării să fie nu doar interesantă din punct de vedere teoretic ci importantă şi din punct de vedere practic.

Generalizarea pleacă de la noţiunea de *partiţie fuzzy* (vezi Capitolul 2). În loc să spunem că un obiect aparţine unei anumite clase constituente a unei partiţii atribuim un grad de apartenenţă $A_{ij} = A_i(\mathbf{x}^j) \in [0, 1]$ astfel înc"at să fie respectată condiţia de normare:

$$\sum_{i=1}^{n} A_{ij} = \sum_{i=1}^{n} A_i(\mathbf{x}^j) = 1 \tag{3.39}$$

Plecînd de la aceste ipoteze pentru fiecare caz în parte vom redefini măsurile de disimilaritate, expresia funcţiei criteriu şi algoritmii de minimizare a acesteia.

3.3.1 Algoritmul n-medii fuzzy

Reconsiderăm problema clasificării cu clusteri sferici în lumina ipotezelor mai sus enunţate.

Măsura de disimilaritate pe care vrem să o introducem vrem să păstreze din caracteristicile măsurii de disimilaritate din cazul clasic dar care să ţină seama şi de gradul de apartenenţă. Introducem o nouă metrică d_i indusă de metrica euclidiană $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$ după cum urmează:

$$d_i(\mathbf{x}, \mathbf{y}) = \begin{cases} \min(A_i(\mathbf{x}), A_i(\mathbf{y}))d(\mathbf{x}, \mathbf{y}) & , \ \mathbf{x}, \mathbf{y} \in \mathbf{X} \\ A_i(\mathbf{x})d(\mathbf{x}, \mathbf{y}) & , \ \mathbf{x} \in \mathbf{X}, \ \mathbf{y} \notin \mathbf{X} \\ d(\mathbf{x}, \mathbf{y}) & , \ \mathbf{x}, \mathbf{y} \notin \mathbf{X} \end{cases} \tag{3.40}$$

Avem nevoie de distanţa d_i pentru a defini inadecvarea dintre un element al clasei $\mathbf{X}$ şi un prototip din $\mathbf{L}$:

$$D(\mathbf{x}^j, \mathbf{L}^i) = (d_i(\mathbf{x}^j, \mathbf{L}_i))^2 \tag{3.41}$$

Prototipurile sunt tot puncte din spaţiul atributelor (spaţiul euclidian d dimensional).

Ţinînd cont că $\mathbf{x}$ din 3.41 aparţine totdeauna mulţimii $\mathbf{X}$ şi că dacă $\mathbf{L}_i$ este un element al acestei mulţimi, gradul său de apartenenţa trebuie să fie mai mare dec"at al oricărui alt element (în definitiv este elementul care reprezintă cel mai bine o clasă deci e normal să aibă cel mai mare grad de apartenenţa la aceasta) putem deduce că:

$$d_i(\mathbf{x}^j, \mathbf{L}_i) = A_i(\mathbf{x}^j)d(\mathbf{x}^j, \mathbf{L}_i) \tag{3.42}$$

şi deci măsura de disimilaritate va fi:

$$D_i(\mathbf{x}^j, \mathbf{L}^i) = (d_i(\mathbf{x}^j, \mathbf{L}_i))^2 = A_i^2(\mathbf{x}^j)d^2(\mathbf{x}^j, \mathbf{L}_i) \tag{3.43}$$

Măsura inadecvării reprezentării clasei A_i prin prototipul $\mathbf{L}_i$ este:

$$I(A_i, \mathbf{L}_i) = \sum_{j=1}^{p} D_i(\mathbf{x}^j, \mathbf{L}_i) = \sum_{j=1}^{p} A_i^2(\mathbf{x}^j)d^2(\mathbf{x}^j, \mathbf{L}_i) \tag{3.44}$$

Observaţie: Gradele de apartenenţă ale elementelor mulţimii $\mathbf{X}$ la clasele A_i pot fi reprezentate printr-o matrice cu n linii şi p coloane:

$$A_{ij} = A_i(\mathbf{x}^j) \in [0, 1], \ i = 1..n, \ j = 1..p \tag{3.45}$$

Inadecvarea reprezentării partiţiei nuanţate $\mathbf{P} = \{A_1, ..., A_n\}$ prin $\mathbf{L} = \{\mathbf{L}_1, ..., \mathbf{L}_n\}$ este dată de funcţia criteriu:

$$J(\mathbf{P}, \mathbf{L}) = \sum_{i=1}^{n} \sum_{j=1}^{p} A_i^2(\mathbf{x}^j)d^2(\mathbf{x}^j, \mathbf{L}_i) \tag{3.46}$$

Problema de clasificare fuzzy revine la determinarea partiţiei nuanţate $\mathbf{P}$ şi a reprezentării acesteia care realizează minimizarea acestei funcţii criteriu.

În acest raţionament nu am ţinut seama de forma distanţei d şi deci rezultatul de mai sus este valabil indiferent de aceasta. De fapt îl vom folosi ca punct de plecare pentru toţi algoritmii de clasificare fuzzy.

Particularizăm acum formula 3.46 pentru clusteri sferici. Ţinînd cont că $d(\mathbf{x}^j, \mathbf{L}_i) = \|\mathbf{x}^j, \mathbf{L}_i\|$ putem transcrie 3.46 în felul următor:

$$J(\mathbf{P}, \mathbf{L}) = \sum_{i=1}^{n} \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) \left\|\mathbf{x}^j, \mathbf{L}_i\right\|^2 \tag{3.47}$$

sau luînd în considerare expresia produsului scalar în spaţii euclidiene:

$$J(\mathbf{P}, \mathbf{L}) = \sum_{i=1}^{n} \sum_{j=1}^{p} A_i^2(\mathbf{x}^j)(\mathbf{x}^j - \mathbf{L}_i)^\top(\mathbf{x}^j - \mathbf{L}_i) \tag{3.48}$$

Minimizarea analitică a acestei funcţii nefiind posibilă dezvoltăm o metodă iterativă asemănătoare cu cea de la cazul clasic, metodă care constă dintr-o succesiune de paşi Picard.

Minimul funcţiei $J(., \mathbf{L})$ e realizat de partiţia nuanţată $\mathbf{P} = \{A_1, ..., A_n\}$ cu:

$$A_i(\mathbf{x}^j) = \frac{1}{\displaystyle\sum_{k=1}^{n} \frac{d^2(\mathbf{x}^j, \mathbf{L}_i)}{d^2(\mathbf{x}^j, \mathbf{L}_k)}}, \ j = 1..p, \ i = 1..n \tag{3.49}$$

Minimul funcţiei $J(\mathbf{P}, .)$ e realizat de reprezentarea $\mathbf{L} = \{\mathbf{L}_1, ..., \mathbf{L}_n\}$ cu:

$$\mathbf{L}_i = \frac{\displaystyle\sum_{j=1}^{p} A_i^2(\mathbf{x}^j)\mathbf{x}^j}{\displaystyle\sum_{k=1}^{p} A_i^2(\mathbf{x}^j)}, \ i = 1..n \tag{3.50}$$

Pentru o demonstraţie a acestor formule vezi [9] pag. 99–102.

Folosind aceste rezultate putem scrie Algoritmul 4. Spre deosebire de algoritmul clasic, procesul iterativ converge doar spre soluţia exactă, neatingînd-o însă niciodată. Criteriul de oprire a algoritmului folosit în celălalt caz nu mai funcţionează deci. Putem opri procesul iterativ c"and modificările aduse de ultimul pas sunt minore. Pentru a exprima matematic această idee ne folosim de o normă pe matrici de exemplu $\|A\| = \max|A_{ij}|$. Considerăm că procesul iterativ s-a încheiat atunci c"and $\left\|\mathbf{P}^l - \mathbf{P}^{l+1}\right\| < \varepsilon$.

Algoritmul 4 Algoritmul n-medii fuzzy

Require: $\mathbf{P}^0 = \{A_1, A_2, ..., A_n\}$ o partiţie fuzzy iniţială a lui $\mathbf{X}$ descrisă de matricea A

Ensure: O partiţie care minimizează funcţia J

repeat

  for all mulţimile A_i ce formează partiţia $\mathbf{P}$ **do**

    Se calculează prototipul mulţimii A_i cu formula:

$$\mathbf{L}_i = \frac{\sum_{j=1}^{p} A_i^2(\mathbf{x}^j)\mathbf{x}^j}{\sum_{k=1}^{p} A_i^2(\mathbf{x}^j)}, \ i = 1..n$$

    Se recalculează gradele de apartenenţă la clasa A_i cu formula:

$$A_i(\mathbf{x}^j) = \frac{1}{\sum_{k=1}^{n} \frac{d^2(\mathbf{x}^j, \mathbf{L}_i)}{d^2(\mathbf{x}^j, \mathbf{L}_k)}}, \ j = 1..p, \ i = 1..n$$

  end for

until $\left\|\mathbf{P}^l - \mathbf{P}^{l+1}\right\| < \varepsilon$

Conform acestui algoritm avem nevoie de un spaţiu de memorie de $2 \times n \times p$ şi de un algoritm de ordinul $\mathcal{O}(n^2)$ pentru a calcula $\left\|\mathbf{P}^l - \mathbf{P}^{l+1}\right\|$. Dacă volumul datelor este considerabil de mare acest lucru poate deveni o problemă.

Vom face c"ateva observaţii care sa ne ajute să optimizăm algoritmul 4. Pentru început să observăm că în momentul în care algoritmul a convers în matricea A şi în reprezentarea $\mathbf{L}$ se află aceiaşi informaţie (fiecare putîndu-se deduce din cealaltă cu formulele 3.49 şi 3.50) şi deci în final avem nevoie doar de una dintre ele.

Algoritmul 5 Algoritmul n-medii fuzzy—varianta optimizată

Require: $\mathbf{L}$ o reprezentare iniţială

Ensure: O reprezentare care minimizează funcţia J

repeat
  $\mathcal{S}_{A,i} = 0,\ \mathcal{S}_{\mathbf{x},i} = 0$ {iniţializarea datelor}
  for all $\mathbf{x}^j \in \mathbf{X}$ **do**
    $\mathcal{C} = \sum_{i=1}^{n} \frac{1}{d^2(\mathbf{x}^j, \mathbf{L}_i)}$
    for all $A_i \in \mathbf{P}$ **do**
      $\mathcal{A}_i = \frac{1}{\mathcal{C} d^2(\mathbf{x}^j, \mathbf{L}_i)}$
      $\mathcal{S}_{\mathbf{x},i} = \mathcal{S}_{\mathbf{x},i} + \mathcal{A}_i^2 \mathbf{x}$
      $\mathcal{S}_{A,i} = \mathcal{S}_{A,i} + \mathcal{A}_i^2$
    end for
  end for
  for all $A_i \in \mathbf{P}$ **do**
    $\mathbf{L}_i = \frac{\mathcal{S}_{\mathbf{x},i}}{\mathcal{S}_{A,i}}$
  end for
until $\left\|\mathbf{L}^l - \mathbf{L}^{l+1}\right\| < \varepsilon$

Din punct de vedere al spaţiului de memorare (avantaje pot apărea şi la algoritmii de procesare) este de preferat memorarea doar a reprezentării. Apare o mică problemă care însă nu e fundamentală şi poate fi uşor rezolvată: pentru a opri algoritmul am folosit o normă pe o diferenţă de matrici cu grade de apartenenţă. Pentru a calcula norma avem nevoie de ambele matrici şi tocmai memorarea acestora vrem să o evităm. Însă metoda de oprire a algoritmului nu este unică posibilă şi nu sunt motive teoretice care să ne facă să credem că un criteriu bazat pe matricile de apartenenţă e mai bun dec"at un criteriu bazat pe reprezentarea claselor. Putem deci opri algoritmul în momentul în care reprezentarea nu se schimbă prea mult (scris matematic $\left\|\mathbf{L}^l, \mathbf{L}^{l+1}\right\| < \varepsilon$).

Un alt aspect se referă la începerea procesului iterativ. În loc să plecăm de la o partiţie iniţială putem pleca de la o reprezentare iniţială. De fapt putem avea un avantaj important procedînd astfel. Plecînd de la o partiţie iniţială algoritmul are nevoie de c"ateva iteraţii pentru a grupa datele în nori pe c"and dacă se pleacă de la o reprezentare iniţială datele sunt deja grupate. Pe de altă parte e greu de ales o partiţie iniţială care să fie c"at de c"at avantajoasă fiind însă uşor de ales o reprezentare iniţială avantajoasă (se aleg de exemplu ca prototipuri iniţiale n obiecte din $\mathbf{X}$, o discuţie detaliată vom face în Capitolul 8).

Cu puţin efort operaţiile necesare poată fi în aşa fel aranjate înc"at cu preţul unui spaţiu minim de memorie suplimentar să se facă un număr relativ mic de operaţii elementare. Algoritmul care respectă toate amendamentele enunţate mai sus este prezentat în Algoritmul 5

3.3.2 Clasificare fuzzy cu prototipuri varietăţi liniare

Reluăm problema clasificării cu prototipuri varietăţi liniare expusă în Secţiunea 3.2.5 din perspectiva logicii fuzzy în spiritul secţiunii precedente.

Prototipurile claselor pe care le luăm în considerare sunt varietăţi liniare (generalizări ale dreptei şi planului în spaţii d dimensionale).

Reamintim că distanţa de la un punct $\mathbf{x}^j$ la un prototip varietate liniară $\mathbf{L}_i$ descris de un punct $\mathbf{v}^i$ şi m direcţii $\mathbf{u}^{1i}, ..., \mathbf{u}^{mi}$ este:

$$d(\mathbf{x}^j, \mathbf{L}_i) = sqrt\left\|\mathbf{x}^j - \mathbf{v}^i\right\|^2 - \sum_{k=1}^{m}(\mathbf{x}^j - \mathbf{v}^i, \mathbf{u}^{ki})^2 \tag{3.51}$$

Înlocuind în expresia generală a funcţiei criteriu 3.46 obţinem:

$$J(\mathbf{P}, \mathbf{L}) = \sum_{i=1}^{n}\sum_{j=1}^{p} A_i^2(\mathbf{x}^j)(\left\|\mathbf{x}^j - \mathbf{v}^i\right\|^2 - \sum_{k=1}^{m}(\mathbf{x}^j - \mathbf{v}^i, \mathbf{u}^{ki})^2) \tag{3.52}$$

Minimizarea ecuaţiei 3.52 se face tot printr-o metodă iterativă constînd într-o succesiune de paşi Picard. Următoarele rezultate a căror demonstraţie se găseşte în [9] servesc la construirea algoritmului de clasificare fuzzy cu prototipuri varietăţi liniare.

$\mathbf{P}$ este un minim al funcţiei $J(., \mathbf{L})$ dacă şi numai dacă:

$$A_i(\mathbf{x}^j) = \frac{1}{\sum_{k=1}^{n} \frac{d^2(\mathbf{x}^j, \mathbf{L}_i)}{d^2(\mathbf{x}^j, \mathbf{L}_k)}}, \ j = 1..p, \ i = 1..n \tag{3.53}$$

Algoritmul 6 Clasificare fuzzy cu prototipuri varietăţi liniare

Require: $\mathbf{P}^0 = \{A_1, A_2, ..., A_n\}$ o partiţie fuzzy iniţială a lui $\mathbf{X}$ descrisă de matricea A

Ensure: O partiţie care minimizează funcţia J

1: **repeat**

2: **for all** mulţimile A_i ce formează partiţia **P do**

3: Se calculează prototipul mulţimii A_i cu formulele:

$$\mathbf{v}_i = \frac{\sum_{j=1}^{p} A_i^2(\mathbf{x}^j)\mathbf{x}^j}{\sum_{k=1}^{p} A_i^2(\mathbf{x}^j)},\ i = 1..n$$

$\mathbf{u}^{1i}, ..., \mathbf{u}^{mi}$ sunt vectorii proprii corespunzători celor mai mari m valori proprii ai matricii

$$S_i = \sum_{j=1}^{p} A_i^2(\mathbf{x}^j)(\mathbf{x}^j - \mathbf{v}^i)^\top(\mathbf{x}^j - \mathbf{v}^i)$$

4: Se recalculează gradele de apartenenţă la clasa A_i cu formula:

$$A_i(\mathbf{x}^j) = \frac{1}{\sum_{k=1}^{n} \frac{d^2(\mathbf{x}^j, \mathbf{L}_i)}{d^2(\mathbf{x}^j, \mathbf{L}_k)}},\ j = 1..p,\ i = 1..n$$

5: **end for**

6: **until** $\left\|\mathbf{P}^l - \mathbf{P}^{l+1}\right\| < \varepsilon$

$\mathbf{L}$ este un minim al funcţiei $J(\mathbf{P}, .)$ dacă şi numai dacă:

$$\mathbf{v}^i == \frac{\sum_{j=1}^{p} A_i^2(\mathbf{x}^j)\mathbf{x}^j}{\sum_{k=1}^{p} A_i^2(\mathbf{x}^j)},\ i = 1..n \tag{3.54}$$

şi $\mathbf{u}^{1i}, ..., \mathbf{u}^{mi}$ sunt vectorii proprii corespunzători celor mai mari m valori proprii ai matricii:

$$S_i = \sum_{j=1}^{p} A_i^2(\mathbf{x}^j)(\mathbf{x}^j - \mathbf{v}^i)^\top(\mathbf{x}^j - \mathbf{v}^i) \tag{3.55}$$

Algoritmul de clasificare fuzzy cu prototipuri varietăţi liniare este prezentat în Algoritmul 6

Ca şi în cazul algoritmului n-medii fuzzy dezvoltăm un algoritm mai avantajos din aceleaşi considerente. Raţionamentul prezentat atunci (vezi Secţiunea 3.3.1) răm"ane valabil şi în această situaţie, algoritmul fiind doar puţin adaptat pentru prototipuri varietăţi liniare.

Există o dificultate suplimentară în proiectarea acestui algoritm vizavi de algoritmul n-medii fuzzy. Această dificultate se referă la calculul matricii S. Dorim să facem acest calcul în maniera calculului centrului de greutate adică calculînd nişte coeficienţi care după o singură trecere peste date să ne furnizeze valoarea căutată. Dar în expresia matricii S_i intră $\mathbf{v}^i$ care se poate calcula doar cu o trecere suplimentară.

În acest moment sunt disponibile mai multe variante. Prima ar fi să calculăm S_i folosind o trecere suplimentară, variantă care nu prea este avantajoasă. O a doua variantă ar fi aproximarea $\mathbf{v}^i$ cu valoarea anterioară care este deja calculată. Cu c"at procesul de clasificare avansează diferenţa dintre valori la paşi succesivi ai vectorului $\mathbf{v}^i$ e din ce în ce mai mică deci modificări esenţiale nu apar. Acest algoritm este prezentat în Algoritmul 7.

Să privim însă cu atenţie la ecuaţia 3.55. Încercăm să o scriem într-o altă formă care poate fi mai avantajoasă. Elementul de coordonate (α, β) al matricii S are expresia:

$$
\begin{aligned}
S_{\alpha\beta} &= \sum_{j=1}^{p} A_i^2(\mathbf{x}^j)(x_\alpha^j - v_\alpha^i)(x_\beta^j - v_\beta^i) \\
&= \sum_{j=1}^{p} A_i^2(\mathbf{x}^j)(x_\alpha^j x_\beta^j - v_\alpha^i x_\beta^j - x_\alpha^j v_\beta^i + v_\alpha^i v_\beta^i) \\
&= \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) x_\alpha^j x_\beta^j - v_\alpha^i \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) x_\beta^j - v_\beta^i \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) x_\alpha^j + v_\alpha^i v_\beta^i \sum_{j=1}^{p} A_i^2(\mathbf{x}^j)
\end{aligned}
$$

Ţinînd seama de notaţiile pentru sume parţiale din Algoritmul 7 şi de ecuaţia 3.54 putem

Algoritmul 7 Clasificare fuzzy cu prototipuri varietăţi liniare—varianta optimizată

Require: $\mathbf{L}$ o reprezentare iniţială

Ensure: O reprezentare care minimizează funcţia J

repeat
    $\mathcal{S}_{A,i} = 0,\ \mathcal{S}_{\mathbf{x},i} = 0\ S_i = 0_{dd}$ {iniţializarea datelor}
    for all $\mathbf{x}^j \in \mathbf{X}$ **do**
        $\mathcal{C} = \sum_{i=1}^{n} \frac{1}{d^2(\mathbf{x}^j, \mathbf{L}_i)}$
        for all $A_i \in \mathbf{P}$ **do**
            $\mathcal{A}_i = \frac{1}{\mathcal{C} d^2(\mathbf{x}^j, \mathbf{L}_i)}$
            $\mathcal{S}_{\mathbf{x},i} = \mathcal{S}_{\mathbf{x},i} + \mathcal{A}_i^2 \mathbf{x}$
            $\mathcal{S}_{A,i} = \mathcal{S}_{A,i} + \mathcal{A}_i^2$
            $S_i = S_i + \mathcal{A}_i^2 (\mathbf{x}^j - \mathbf{v}^i)(\mathbf{x}^j - \mathbf{v}^i)^\top$
        end for
    end for
    for all $A_i \in \mathbf{P}$ **do**
        $\mathbf{L}_i = \frac{\mathcal{S}_{\mathbf{x},i}}{\mathcal{S}_{A,i}}$
        $\mathbf{u}^{1i}, ..., \mathbf{u}^{mi}$ sunt vectorii proprii corespunzători celor mai mari m valori proprii ai matricii S_i
    end for
until $\left\| \mathbf{L}^l - \mathbf{L}^{l+1} \right\| < \varepsilon$

scrie:

$$\mathcal{S}_{A,i} = \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) \tag{3.56}$$

$$\mathcal{S}_{\mathbf{x},i}^{\alpha} = \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) x_\alpha^j \tag{3.57}$$

$$\mathcal{S}_{\mathbf{x},i}^{\beta} = \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) x_\beta^j \tag{3.58}$$

$$v_\alpha^i = \frac{\mathcal{S}_{\mathbf{x},i}^{\alpha}}{\mathcal{S}_{A,i}} \tag{3.59}$$

$$v_\beta^i = \frac{\mathcal{S}_{\mathbf{x},i}^{\beta}}{\mathcal{S}_{A,i}} \tag{3.60}$$

Expresia 3.56 devine:

$$\begin{aligned} S_{\alpha\beta} &= \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) x_\alpha^j x_\beta^j - \frac{\mathcal{S}_{\mathbf{x},i}^{\alpha}\mathcal{S}_{\mathbf{x},i}^{\beta}}{\mathcal{S}_{A,i}} - \frac{\mathcal{S}_{\mathbf{x},i}^{\alpha}\mathcal{S}_{\mathbf{x},i}^{\beta}}{\mathcal{S}_{A,i}} + \frac{\mathcal{S}_{\mathbf{x},i}^{\alpha}\mathcal{S}_{\mathbf{x},i}^{\beta}}{\mathcal{S}_{A,i}} \\ &= \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) x_\alpha^j x_\beta^j + \frac{\mathcal{S}_{\mathbf{x},i}^{\alpha}\mathcal{S}_{\mathbf{x},i}^{\beta}}{\mathcal{S}_{A,i}} \end{aligned}$$

Făcînd notaţia:

$$S_{\mathbf{x},i}^{\alpha\beta} = \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) x_\alpha^j x_\beta^j \tag{3.61}$$

şi înlocuind în 3.61 obţinem:

$$S_{\alpha\beta} = S_{\mathbf{x},i}^{\alpha\beta} + \frac{\mathcal{S}_{\mathbf{x},i}^{\alpha}\mathcal{S}_{\mathbf{x},i}^{\beta}}{\mathcal{S}_{A,i}} \tag{3.62}$$

În virtutea acestor ecuaţii rescriem Algoritmul 7 în Algoritmul 8. Observăm că nu doar că al doilea algoritm nu face aproximaţia de care am pomenit la descrierea primului algoritm dar numărul de operaţii necesare se reduce (facem economie de operaţii de scădere, o discuţie detaliată în legătură cu acest aspect o vom face în capitolele 8 şi 7.

3.3.3 Combinaţii convexe de funcţii criteriu

Clasificarea cu prototipuri sferice este prea săracă (nu prea oferă informaţii legate de forma clusterilor) iar clasificarea cu prototipuri varietăţi liniare are dezavantajul că permite clusterilor să fie oric"at de extinşi (clusteri nelocalizaţi). Ne punem problema combinării celor două tipuri de clasificare.

Algoritmul 8 Clasificare fuzzy cu prototipuri varietăţi liniare—a doua variantă optimizată

Require: $\mathbf{L}$ o reprezentare iniţială

Ensure: O reprezentare care minimizează funcţia J

repeat
    $\mathcal{S}_{A,i} = 0,\ \mathcal{S}_{\mathbf{x},i} = 0\ S_{\mathbf{x},i} = 0_{dd}$ {iniţializarea datelor}
    for all $\mathbf{x}^j \in \mathbf{X}$ **do**
        $\mathcal{C} = \sum_{i=1}^{n} \frac{1}{d^2(\mathbf{x}^j, \mathbf{L}_i)}$
        for all $A_i \in \mathbf{P}$ **do**
            $\mathcal{A}_i = \frac{1}{\mathcal{C} d^2(\mathbf{x}^j, \mathbf{L}_i)}$
            $\mathcal{S}_{\mathbf{x},i} = \mathcal{S}_{\mathbf{x},i} + \mathcal{A}_i^2 \mathbf{x}$
            $\mathcal{S}_{A,i} = \mathcal{S}_{A,i} + \mathcal{A}_i^2$
            $S_{\mathbf{x},i} = S_{\mathbf{x},i} + \mathcal{A}_i^2 \mathbf{x}^j \mathbf{x}^{j\top}$
        end for
    end for
    for all $A_i \in \mathbf{P}$ **do**
        $S_i = S_{\mathbf{x},i} + \frac{\mathcal{S}_{\mathbf{x},i} \mathcal{S}_{\mathbf{x},i}^\top}{\mathcal{S}_{A,i}}$
        $\mathbf{L}_i = \frac{\mathcal{S}_{\mathbf{x},i}}{\mathcal{S}_{A,i}}$
        $\mathbf{u}^{1i}, ..., \mathbf{u}^{mi}$ sunt vectorii proprii corespunzători celor mai mari m valori proprii ai matricii S_i
    end for
until $\left\| \mathbf{L}^l - \mathbf{L}^{l+1} \right\| < \varepsilon$

Numim combinaţie convexă de funcţii criteriu funcţiile de forma:

$$J_a = \sum_{k=1}^{s} a_k J_k \tag{3.63}$$

unde J_k sunt funcţii criteriu de forma celor deja considerate şi a_k respectă condiţia de normare:

$$\sum_{k=1}^{p} a_k = 1 \tag{3.64}$$

Ne punem problema combinării de funcţii criteriu pentru clusteri sferici şi pentru clusteri varietăţi liniare de diferite ordine. Putem considera că clusterii sferici sunt de fapt varietăţi liniare de ordin 0.

Notăm cu $\mathbf{L}^{r_k}$ o varietate liniară de ordin $r_k \in 0, 1, ..., d$. Funcţia criteriu:

$$J(\mathbf{P}, \mathbf{L}^{r_1}, ..., \mathbf{L}^{r_s}) = \sum_{k=1}^{s} a_k J(\mathbf{P}, \mathbf{L}^{r_k}) \tag{3.65}$$

este o combinaţie convexă de funcţii criteriu.

Luînd în considerare formele funcţiilor criteriu putem pune J în forma:

$$J(\mathbf{P}, \mathbf{L}^{r_1}, ..., \mathbf{L}^{r_s}) = \sum_{i=1}^{n} \sum_{j=1}^{p} A_i^2(\mathbf{x}^j) D_{ij} \tag{3.66}$$

unde D_{ij} are forma:

$$D_{ij} = \sum_{k=1}^{s} a_k \left(\left\| \mathbf{x}^j - \mathbf{v}^i \right\|^2 - \sum_{l=1}^{r_k} (\mathbf{x}^j - \mathbf{v}^i, \mathbf{u}^{li}) \right) \tag{3.67}$$

Se pot demonstra (vezi [9] pag. 177–179) doua proprietăţi importante a funcţiilor criteriu de forma 3.65.

> Fixînd partiţia $\mathbf{P}$, $(\mathbf{L}^{r_1}, ..., \mathbf{L}^{r_s})$ este un minim al funcţiei $J(\mathbf{P}, .)$ dacă şi numai dacă oricare ar fi r_k prototipul $\mathbf{L}^{r_k}$ e determinat pe baza prototipului $\mathbf{L}^{r_q}$ unde $r_q = \max r_k$.
>
> Fixînd $(\mathbf{L}^{r_1}, ..., \mathbf{L}^{r_s})$, $\mathbf{P}$ e un minim al funcţiei $J(., \mathbf{L}^{r_1}, ..., \mathbf{L}^{r_s})$ dacă şi numai dacă oricare ar fi A_i aparţinînd partiţiei $\mathbf{P}$ e adevărată ecuaţia:
>
> $$A_i(\mathbf{x}^j) = \frac{1}{\sum_{l=1}^{n} \frac{D_{ij}}{D_{il}}}$$

În virtutea celor spuse mai sus ca algoritm de clasificare mixtă putem folosi oricare din algoritmii prezentaţi în Secţiunea 3.3.2 cu menţiunea că $d^2(\mathbf{x}^j, \mathbf{L}_i)$ se înlocuieşte cu D_{ij} definit cu formula 3.67.

Considerînd cazul particular al combinaţiei dintre o funcţie criteriu sferică şi una pentru varietăţi liniare forma clusterilor obţinuţi este o sferă alungită după anumite direcţii (o formă particulară de elipsoid).

Partea II

Aproximarea funcţiilor

Această secţiune ar putea purta mai multe nume dintre care amintim: *Metode de învăţare*, *Metode de reprezentare a cunoştinţelor*, dar am ales titlul *Aproximarea funcţiilor* deoarece din punct de vedere matematic acest lucru î-şi propun toate metodele enumerate chiar dacă în majoritatea lucrărilor pe această temă nu este pomenit acest lucru.

Aproximarea funcţiilor se referă la determinarea unei funcţii date fiind valori aproximative ale sale în diferite puncte ale domeniului său de definiţie. Funcţia determinată trebuie să fie cea mai bună conform unui anumit criteriu. Cele mai utilizate criterii de cea mai bună potrivire sunt de genul cele mai mici pătrate dar o seamă de alte tipuri de criterii se pot înt"alni în literatură de specialitate.

Problema aproximării funcţiilor are o foarte mare importanţă în stiinţele experimentale, fiind principala metodă de descoperire a legităţilor care caracterizează fenomenele studiate.

Secţiunea conţine trei capitole, fiecare dintre acestea tratînd o metodă de aproximare a funcţiilor. Astfel:

1. Maşinile fuzzy utilizează reprezentări lingvistice şi inferenţa fuzzy pentru a construi modele pentru funcţii şi pentru a utiliza aceste modele

2. Reţelele neuronale folosesc modele bazate pe neuroni aşezaţi în straturi pentru a aproxima funcţii

3. Maşinile bazate pe clusteri sunt o alternativă a autorului la celelalte metode prezentate în această secţiune. Ele se utilizează teoria clusterizării şi au asemănări importante at"at cu maşinile fuzzy c"at şi cu reţelele neuronale reuşind totodată să depăşească neajunsurile acestora, păstrînd totodată calităţile bune.

La listă metodelor enumerate mai sus mai trebuiau adăugate polinoamele de interpolare (Lagrange, Hermite, etc.), polinoamele de cea mai bună aproximare a lui Ceb"aşev şi metoda celor mai mici pătrate. Ţinînd totuşi cont de faptul că ne interesează să aproximăm funcţii neliniare din date afectate de erori, metodele enumerate mai sus nu prea sunt de folos din unul din următoarele motive: sunt prea sensibile la zgomot (erori) sau nu sunt potrivite pentru funcţii neliniare.

Capitolul 4

Maşini fuzzy

Vom trece acum la prezentarea a două tipuri de maşini fuzzy, cele mai des înt"alnite în literatura de specialitate. Vom însoţi prezentarea cu comentarii proprii vizavi de algoritmi şi unde este cazul vom face contrapropuneri la propunerile autorilor.

Înainte de a începe se impune o observaţie. Maşinile fuzzy au apărut şi au fost adesea folosite în cadrul problemelor de control automat. Din acest motiv majoritatea exemplelor şi a terminologiei vine din acest domeniu.

4.1 Maşina fuzzy Mamdani

Primul care a utilizat algoritmi ai logicii fuzzy pentru a construi un controler a fost Mamdani în lucrarea sa *Applications of fuzzy algorithms for control of a simple dynamic plant* ([16]) publicată în 1974. Modelul propus de el este în principiu un aproximator de funcţii, şi constă într-o colecţie de descrieri lingvistice sub formă de implicaţii fuzzy, asemănătoare cu cele prezentate în secţiunea 2.3. Un astfel de model poate fi privit şi ca un sistem expert a cărui bază de cunoştinţe e constituită din reguli fuzzy şi care dispune de un mecanism de inferenţă, în speţă regula generalizată Modus Ponens a lui Zadeh.

4.1.1 Structura generală a unei maşini fuzzy Mamdani

Considerăm o astfel de maşină fuzzy cu m intrări $x_1, x_2, ..., x_m$şi o singură ieşire y care conţine r reguli de forma:

$$L^i : \text{If } x_1 \text{ is } A_{i1} \text{ and...and } x_m \text{ is } A_{im} \text{ then } y \text{ is } B_i \tag{4.1}$$

cu $i = 1..r$. Conform [15] cazul maşinilor cu n ieşiri se reduce la n maşini cu o singură ieşire. În regulile mai sus menţionate avem A_{ij} şi B_i variabile fuzzy.

Mai multe metode de inferenţă au fost propuse pentru maşinile fuzzy Mamdani. Cele mai multe dintre acestea presupun efectuarea a trei paşi:

1. Fuzzyficarea variabilelor de intrare – presupune transformarea valorilor numerice ataşate variabilelor de intrare în valori lingvistice (funcţii de apartenenţă pentru a fi mai exacţi)

2. Utilizarea inferenţei fuzzy (prezentată în secţiunea 2.3) pentru a obţine o valoare lingvistică rezultat plecînd de la sistemul de reguli şi de la valorile lingvistice obţinute prin procedeul de mai sus

3. Defuzzyficarea valorii lingvistice rezultat pentru a obţine valoarea numerică rezultat (metoda centrului de greutate de obicei)

Detalii şi explicaţii suplimentare legate de fiecare etapă în parte pot fi găsite în [15] cap. 4. şi [10] cap. 7.

Alţi autori (vezi [2] şi [1]) au propus metode de inferenţă fuzzy utilizînd reţele neuronale, existînd chear şi implementări hardware a acestor maşini.

Un astfel de model are certe avantaje (vezi şi [10] cap. 7) dintre care amintim:

- Transparenţă – modelul poate fi uşor *citit* de experţi umani
- Uşurinţă în depanare – din aceleaşi motive ca mai sus
- Posibilitatea de interpolare a datelor necunoscute

Totuşi există dezavantaje majore în primul r"and legate de eficienţă. Să aruncăm o scurtă privire asupra mecanismului de inferenţă. Majoritatea autorilor ([15], [10], [1]) acceptă ca şi formulă de inferenţă:

$$\begin{aligned} B'(y) = \max_i \sup_{x_1,\ldots,x_m} & \min(\min(A'_1(x_1), ..., A'_m(x_m)), \\ & \min(A_{i1}(x_1), ..., A_{im}(x_m), B_i(y))) \end{aligned} \tag{4.2}$$

unde $(A'_1...(A'_m$ sunt funcţii de apartenenţă de intrare şi B' e funcţia de apartenenţă de ieşire. Formula de mai sus se poate eventual generaliza folosind norme şi conorme triunghiulare în locul funcţiilor min şi max.

Dacă ca procedeu de fuzzyficare se foloseşte transformarea în punct fuzzy (singleton) atunci procesul de inferenţă se simplifică astfel:

$$B'(y) = \max_i \min(A_{i1}(x_1^0), ..., A_{im}(x_m^0), B_i(y)) \tag{4.3}$$

unde $x_1^0, ..., x_m^0$ sunt valorile numerice care constituie vectorul de intrare.

Pe de altă parte operaţiile min şi max nu pot fi aplicate într-un mod prea elegant asupra funcţiilor analitice se impune ca reprezentarea funcţiilor de apartenenţă să se facă sub forma unei liste de valori în anumite puncte ale universului discursului. De densitatea acestor puncte depinde precizia rezultatului.

Defuzzyficarea (prin metoda centrului de greutate de exemplu) este şi ea foarte greoaie din punct de vedere al efortului de calcul.

Un alt dezavantaj este faptul că ieşirea maşini poate fi mai mare dec"at cel mai mare prototip al vreunei valori lingvistice a variabilei de intrare sau mai mic dec"at cea mai mică. Acest lucru s-ar putea evita dacă rezultatul ar fi o combinaţie convexă a prototipurilor concluziilor regulilor. Din experienţa avută cu produsul MEDISIF autorul a observat că acest fenomen nu este deloc rar, fiind mai degrabă regula dec"at excepţia.

Conform celor discutate mai sus procesul de inferenţă se dovedeşte destul de greoi. Pe de asupra motivaţiile teoretice pentru un astfel de proces de inferenţă (vezi şi secţiunea 2.3) nu sunt destul de clare.

Deşi în literatura de specialitate nu este menţionată, putem să dăm următoarea explicaţie procesului de inferenţă menţionat mai sus: rezultatul procesului de inferenţă (funcţia de apartenenţă rezultat) este o combinaţie a funcţiilor de apartenenţă a consecventului fiecărei reguli, ponderea fiecărei reguli fiind dictată de gradul în care datele de intrare se potrivesc cu antecedentul $(\min(A_{i1}(x_1^0), ..., A_{im}(x_m^0)))$. Această interpretare este evidentă dacă se urmăreşte reprezentarea grafică a procesului de inferenţă din [15] fig. 4.18. Pentru prima dată (deşi într-o formă puţin schimbată) ea a fost propusă de autor în [7].

Dacă presupunem că variabilele lingvistice sunt construite av"and în minte un anumit prototip (presupunere c"at se poate de rezonabilă după cum am arătat în secţiunea 2.3.2), de faptul că procesul de fuzzyficare împreună cu prima parte din inferenţă poate fi privit ca un procedeu de *matching* (compatibilitate intrare–prototip) şi că defuzzyficarea urmăreşte găsirea unui element care să reprezinte cel mai bine funcţia de apartenenţă rezultat (prototipul acesteia) putem da următoarea interpretare a procesului de inferenţă:

- regulile reprezintă comportări tipice (partea de consecvent) în situaţii tipice (partea de antecedent
- procesul de inferenţă deduce modul de comportare în situaţiile normale (atipice) în funcţie de comportările din situaţiile tipice ţinînd cont de gradul de asemănare cu fiecare situaţie tipică.

4.1.2 Identificarea maşinilor fuzzy Mamdani

C"ateva metode de identificare (determinare din exemple de intrare–ieşire) au fost propuse pentru astfel de maşini. Pentru o sinteză a lor vezi [15] cap. 3. Din păcate procedurile de identificare sunt pur euristice, fără un suport teoretic riguros şi fără a lăsa posibilitatea de a impune anumite performanţe rezultatului dorit.

O alta problemă legată de identificarea acestor maşini este faptul ca au un mare număr de parametrii, parametrii greu de determinat deoarece influenţa lor asupra rezultatului final este greu de stabilit (pentru a folosi metode de tipul delta utilizate în cazul antrenării reţelelor neuronale).

Cea mai performantă dintre procedurile de identificare este considerată metoda lui Pedrycz (pentru o descriere a algoritmului vezi [15] alg. A.3.2). Acest algoritm construieşte maşina fuzzy plecînd de la un set de valori lingvistice. În [15] se sugerează ca metodă de determinare a acestor valori lingvistice utilizarea teoriei clusterizării. Aceasta este o idee destul de interesantă dar care din păcate nu utilizează la maximum rezultatele furnizate de teoria clusterizării. În capitolul 6 vom dezvolta algoritmi care fac acest lucru. Pe de altă parte determinarea valorilor lingvistice se poate face concomitent cu determinarea regulilor propriuzise şi deci un astfel de algoritm nu mai este necesar.

Parte din concluziile acestei secţiuni se datorează experienţei autorului, experienţă dob"andită îndeosebi în colaborarea cu Conf. Dr. ing. Dorin Isoc în cadrul proiectului MEDISIF.

Exemple detaliate de utilizare a maşinilor fuzzy Mamdani se pot găsi în [15] şi [10].

4.2 Maşina fuzzy Sugeno

În 1985 Takagi şi Sugeno (vezi [20]) propuneau o variaţie a maşinilor fuzzy Mamdani, variaţie care deşi minoră în formă introduce modificări esenţiale în interpretare.

4.2.1 Structura maşinilor fuzzy Sugeno

Modificarea propusă de cei doi constă în adoptarea ca şi consecvent a regulilor fuzzy a unei expresi liniare a ieşirii faţă de intrări, înlocuind astfel consecventul fuzzy. Regulile sunt astfel de forma:

$$L^i : \text{If } x_1 \text{ is } A_{i1} \text{ and...and } x_m \text{ is } A_{im} \text{ then } y^i = c_{i0} + c_{i1}x_1 + ... + c_{im}x_m \tag{4.4}$$

unde $x_1, ..., x_m$, $A_{i1}, ..., A_{im}$ şi y au semnificaţia din secţiunea anterioară iar $c_{i0}, ..., c_{im}$ sunt constante reale.

După cum se poate observa consecventul regulii este ecuaţia unui hiperplan în spaţiul intrare–ieşire.

Cei doi în articolul mai sus menţionat propuneau următoarea formulă de inferenţă:

$$y = \frac{\sum_{i=1}^{r} w^i y^i}{\sum_{i=1}^{r} w^i} \tag{4.5}$$

unde

$$w^i = \prod_{k=1}^{m} A_{ik}(x_k) \tag{4.6}$$

reprezintă ponderea regulii i în rezultatul final.

Se poate observa că interpretarea dată de către noi maşinilor fuzzy Mamdani se potriveşte şi în acest caz, ba mai mult este chear mult mai uşor de sesizat acest lucru în cazul acestor maşini faţă de cazul precedent. În acest caz însă comportarea tipică sugerată în situaţia tipică este liniară şi nu constantă ca în cazul precedent. Acest lucru ne duce cu g"andul la liniaritatea pe porţiuni (locală) at"at de utilizată în problemele de fizică. Acest lucru a fost sesizat şi de Filev (vezi [13]).

În concluzie acest tip de maşină încearcă să aproximeze o funcţie neliniară printr'o compunere fuzzy de fragmente liniare care aproximează local funcţia neliniară.

Deşi introduce un număr mai mare de parametrii, maşina fuzzy Sugeno aduce c"ateva clarificări care pot duce la simplificarea algoritmului de învăţare. De exemplu partea de inferenţă fuzzy se reduce doar la determinarea ponderilor w^i, coeficienţii hiperplanelor din consecvent putînd fi determinaţi cu o variantă a metodei celor mai mici pătrate.

Faţă de maşinile fuzzy Mamdani, maşinile fuzzy Sugeno au dezavantajul că nu pot fi construite intuitiv (plecînd de la descrieri lingvistice ale experţilor umani) deoarece rareori experţii umani folosesc în consecvent comportări liniare. Un algoritm de *învăţare* a acestor maşini este deci strict necesar. Dacă această identificare se face destul de îngrijit aceste maşini ar trebui să aibă performanţe de aproximare mult superioare maşinilor fuzzy Mamdani (asemănător cu saltul calitativ de la aproximarea unei funcţii cu plafoane constante la aproximarea cu drepte).

4.2.2 Identificarea maşinilor fuzzy Sugeno

În [18] Sugeno şi Kang propun o primă metodă de învăţare, metodă care în principiu constă din următoarele etape:

1. alegerea structurii premizei (antecedentului) – metodă euristică care î-şi propune să determine o partiţionare optimă a spaţiului de intrare astfel ca modelul identificat să fie o c"at mai adecvată reprezentare a sistemului de identificat.

   Se pleacă de la premiza că structura optimă minimizează un criteriu (UC în cazul de faţă).

2. parametrii structurii sunt ajustaţi. Funcţiile de apartenenţă din premisele regulilor sunt de o formă foarte simplă, formă utilizată foarte des în sistemele cu logică fuzzy, şi anume sub formă de trapeze cu baza mare pe linia de 0 şi baza mică pe cea de 1 (vezi [15] pentru detalii). Pentru corectarea parametrilor se foloseşte regula delta (modificări proporţionale cu eroarea).

3. alegerea unei structuri pentru consecvent adică a variabilelor de intrare care intră în expresia ieşirii

4. identificarea parametrilor consecventului adică a constantelor c. Acest lucru se poate face cu metode de tipul celor mai mici pătrate

Să aruncăm o privire şi asupra criteriului UC folosit (preluat din [18]. Mai înt"ai se împart datele observate în două mulţimi N_A cu n_A elemente şi N_B cu n_B elemente. Avem:

$$\mathrm{UC} = \sqrt{\sum_{i=1}^{n_A}(y_i^{AB} - y_i^{AA})^2 + \sum_{i=1}^{n_B}(y_i^{BA} - y_i^{BB})^2} \tag{4.7}$$

unde y_i^{AA} este ieşirea estimată folosind mulţimea N_A ca date de test cu modelul identificat folosind mulţimea N_A ca date de antrenament. La fel sunt definite şi ieşirile y_i^{AB}, y_i^{BA} şi y_i^{BB}.

După cum se poate observa pentru fiecare utilizare a criteriului UC este nevoie de două identificări de modele şi de patru utilizări ale modelului pentru a face predicţii. Evaluarea valorii UC după cum se poate observa din algoritmul descris mai sus este necesară după fiecare modificare de structură, numărul de modificări fiind destul de mare chear şi pentru cazuri simple.

În concluzie algoritmul de mai sus este costisitor şi pe de asupra greu de automatizat, fiind necesară intervenţia unui expert în procesul de învăţare.

Sugeno şi Tanaka au rafinat ulterior ([19]) algoritmul de mai sus însă deficienţele acestuia nu au fost complet îndepărtate, algoritmul răm"anînd unul greoi şi costisitor din punct de vedere computaţional.

Un alt algoritm de antrenare a maşinilor fuzzy Sugeno este propus de Chelaru şi Dumitraşcu în [4]. Cei doi modifică puţin structura maşini propunînd reguli de următoarea formă:

$$L^i : \text{If } \mathbf{x} \text{ is } \mathbf{x}_i \text{ then } y_i = y_{i0} + \sum_{k=1}^{m} c_{ik}(x_k - x_{ik}) \tag{4.8}$$

şi numesc acest tip de maşină sistem fuzzy Sugeno diferenţial.

Mecanismul de inferenţă folosit este următorul:

$$y = \frac{\sum_{i=1}^{r} y_i w^i}{\sum_{i=1}^{r} w^i} \tag{4.9}$$

$$w^i = \frac{1}{\exp\left[\sum_{k=1}^{m} \frac{(x_k - x_{ik})^2}{d_k^2}\right]} \tag{4.10}$$

unde $\mathbf{x}_i = (x_{i1}, ..., x_{im})$ este vectorul de referinţă a reguli i şi d_i este pasul grilei de distribuire a vectorilor $\mathbf{x}_i$ pentru coordonata i. După cum se poate observa antecedentul regulilor este fix (distribuire uniformă în spaţiul intrărilor), doar consecventul răm"anînd de identificat la învăţarea unei asemenea maşini.

Ca metodă de învăţare se sugerează regula delta de la reţele neuronale. Parametrii pasului $l + 1$ se exprimă în funcţie de cei de la pasul l astfel:

$$y_{i0}(l + 1) = y_{i0}(l) + \eta(l)[d(l) - y(l)]w^i(l) \tag{4.11}$$

$$c_{ik}(l + 1) = c_{ik}(l) + \eta(l)[d(k) - y(k)][x_i(l) - x_{ik}(l)]w^i(k) \tag{4.12}$$

unde $d(l)$ este ieşirea dorită la pasul l şi $y(l)$ este ieşirea modelului la pasul l.

Autorii sugerează că un algoritm de clusterizare ar putea fi folosit pentru a determina vectorii $(x)_i$.

După cum vom arăta în capitolul 6 această arhitectură este foarte asemănătoare cu o reţea neuronală radială.

Metoda folosită în acest caz (regula delta) este greu de aplicat în cazul general al maşinilor fuzzy deoarece e greu de calculat derivata erorii faţă de fiecare parametru.

Există un rezultat remarcabil vizavi de maşinile fuzzy Sugeno datorat lui Buckley ([3]), rezultat care pune acest tip de aproximator pe picior de egalitate cu reţelele neuronale. Conform acestuia oricare ar fi o funcţie continuă există o maşină fuzzy Sugeno care să aproximeze această funcţie cu orice grad de acurateţe.

Maşinile fuzzy Sugeno au evidente avantaje vizavi de maşinile fuzzy Mamdani. La aceste avantaje se adaugă faptul că stabilitatea sistemelor în care intră aceste maşini poate fi studiată cu destul de multă uşurinţă (vezi [21] şi [5]).

Capitolul 5

Reţele neuronale

5.1 Introducere

Reţelele neuronale au fost intens studiate în ultimul deceniu, făcînd obiectul de studiu a numeroase lucrări ştiinţifice. Astăzi ele constituie un capitol aparte al *Inteligenţei artificiale*.

Istoria lor deşi scurtă ca întindere temporară (doar 50 ani) este bogată în evenimente. O frumoasă relatare a acestei istorii este prezentată de Adriana Dumitraş în [8].

Diversitatea tipurilor de reţele neuronale propuse p"ană în prezent este aşa de mare că o expunere exhaustivă a acestora ar cuprinde probabil mai multe volume. Din acest motiv vom prezenta doar c"ateva principii legate de două tipuri de reţele care sunt importante pentru demersul nostru, în speţă reţelele neuronale feedforward şi reţelele neuronale radiale.

Descrieri detaliate legate de reţelele neuronale pot fi găsite în [8], [12] şi [11]. Scopul prezentării din capitolul de faţă este legat de susţinerea unora din ideile prezentate în capitolul 6.

Prezentarea va fi în primul r"and informativă interesîndu-ne în primul r"and aspecte calitative legate de reţelele neuronale şi nu o prezentare riguroasă a acestora.

5.2 Reţele neuronale feedforward cu neuroni simpli

În accepţiunea majorităţi comunităţii ştiinţifice reţelele neuronale feedforward sunt formate din straturi succesive de neuroni, neuronii fiecărui strat trimiţînd informaţii doar neuronilor din stratul imediat următor, astfel informaţia circulă doar într-un singur sens (de aici şi sintagma feedforward în nume).

5.2.1 Structura reţelelor neuronale feedforward

'In structurile neuronale de acest tip toţi neuronii sunt identici indiferent de stratul din care fac parte. Un neuron are un număr de intrări neuronale (analogul dendritelor) şi o singură ieşire (analogul axonului). Expresia ieşirii tipului de neuron cel mai popular (datorat lui McCulloch şi Pitts) este următoarea:

$$y = f(net) = f(\sum_{i=1}^{m} w_i x_i + b) \tag{5.1}$$

unde f este funcţia de activare continuă, derivabilă şi eventual monotonă (sigmoida de exemplu), x_i este intrarea i şi w_i este ponderea acesteia. După cum se poate observa neuronul se comportă ca şi un sumator.

Alegerea funcţiei f cu proprietăţile mai sus enumerate este importantă pentru ca să se poată construi o metodă de antrenare pentru reţelele neuronale care au în componenţă astfel de neuroni. O altă proprietate importantă a funcţiei de activare este neliniaritatea acesteia, dat fiind faptul că o reţea neuronală cu funcţie de activare liniară poate învăţa doar date linear separabile (vezi [12] sec. 3.7 pentru o discuţie detaliată asupra acestui aspect).

5.2.2 Învăţarea reţelelor neuronale feedforward

Învăţarea în structurile de tip *feedforward* poate fi privită ca un proces de aproximare. Există c"ateva rezultate extraordinare legate de capacităţile de aproximare ale acestor tipuri de reţele neuronale. Astfel se poate arăta că reţelele neuronale feedforward sunt aproximatori universali ceea ce înseamnă că oricare ar fi o funcţie continuă există o reţea neuronală cu un strat intermediar (trei straturi deci, primul fiind de intrare şi ultimul de ieşire) care poate aproxima oric"at de bine această funcţie. O discuţie detaliată asupra acestui aspect se poate găsi în [8].

Totodată învăţarea reţelelor neuronale poate fi privită şi ca proces de optimizare, învăţarea constînd în minimizarea unei funcţii cost (de obicei pătratică).

O metodă foarte des folosită de optimizare a funcţiei cost este metoda gradientului care constă în modificarea parametrilor modelului în direcţia gradientului. Această metodă poate fi privită şi ca propagarea erorii de învăţare de la ieşire înspre intrare, procedeu numit *backpropagation*.

Algoritmul 9 este un algoritm de învăţare pentru reţelele neuronale feedforward adaptat după algoritmii prezentaţi în [22]. Pasul forward poate fi folosit pentru utilizarea reţelei după ce aceasta a fost antrenată.

Metode de învăţare mai avansate pot fi găsite în literatură (vezi [8] de ex.). La acest tip de reţea apar o seamă de probleme:

- reţeaua converge într-un minim local depărtat de minimul global al funcţiei criteriu
- reţeaua se blochează nemaiputîndu-se învăţa ceva
- reţeaua învaţă prea bine datele de antrenament şi găseşte reguli aleatoare în date astfel că, deşi eroarea de reproducere a datelor de antrenament e mică, eroarea de reproducere a unui alt set de date (date de test) distribuite în spaţiul intrărilor la fel ca datele de antrenament este mare. Se spune că reţeaua nu generalizează.
- dimensiunea reţelei pentru o problemă dată este greu de stabilit şi se face de obicei printr-o metodă euristică
- nu se poate da nici o interpretare fizică valorilor ponderilor fiind imposibil de depanat o astfel de reţea
- este nevoie de o funcţie neliniară care în general este costisitoare din punct de vedere al timpului de calcul (echivalentă cu zeci de operaţii elementare)

5.3 Reţele neuronale radiale

Reţelele neuronale radiale sunt (cel puţin în varianta lor clasică) reţele neuronale cu exact trei straturi: unul de intrare, unul intermediar format din neuroni cu funcţie de activare radială şi unul de ieşire care face doar sumare.

După cum se observă în acest tip de reţea neuronii din stratul intermediar sunt diferiţi de cei din stratul de ieşire. Funcţia de activare a acestora este de forma:

$$f(\mathbf{x}) = \varphi(\|\mathbf{x} - \mathbf{x}_i^c\|) \tag{5.2}$$

unde $\mathbf{x}$ e vectorul de intrare (se observă că intrarea este preluată fără ponderare în stratul intermediar), $\mathbf{x}_i^c$ sunt centrii funcţiei radiale. Se observă că funcţia de activare are simetrie sferică.

Algoritmul 9 Algoritmul backpropagation

Require: $(\mathbf{x}_k, \mathbf{y}_k)$ un set de date de antrenament

Ensure: reţea neuronală antrenată (eroare medie $< \epsilon$)

repeat
  for all perechile de intrare-ieşire $(\mathbf{x}_k, \mathbf{y}_k)$ **do**
    conectează intrarea $\mathbf{x}_k$ la intrarea primului strat
    for all straturi de la intrare la ieşire **do**
      for all Neuroni în stratul curent **do**
        $o = f(\mathbf{wi} + b)$ {$\mathbf{w}$ e vectorul ponderilor, b e pragul neuronului, $\mathbf{i}$ vectorul intrare în neuron, o ieşirea neuronului}
      end for
      conectează ieşirea acestui strat la intrarea stratului următor
    end for
    Calculează eroarea la ieşire ca fiind diferenţa dintre ieşirea prezisă (vectorul o al ultimului strat) şi cea adevărată. Aceasta va fi eroarea la intrare a ultimului strat
    for all straturi de la ieşire la intrare **do**
      for all Neuroni în stratul curent **do**

$$\delta = \mathrm{err}_{\mathrm{in}} f'(o)$$

$$\mathbf{w} = \mathbf{w} + \eta\delta\mathbf{i}$$

$$b = b + \eta\delta$$

        {$\mathrm{err}_{\mathrm{in}}$ e eroarea la intrare, η e coeficientul de învăţare}
      end for

$$\mathbf{err}_{\mathrm{out}} = (\delta \mathbf{w} f'(\mathbf{o})$$

      pune eroarea la ieşire ca eroare la intrare în următorul strat
    end for
  end for
until Eroarea media pătratică $< \epsilon$

Cea mai folosită funcţie φ este funcţia gausiană (reţelele care folosesc o astfel de funcţie se numesc *reţele neuronale gausiene*.

$$\varphi(r) = \frac{1}{e^{-\frac{r^2}{2\sigma^2}}} \tag{5.3}$$

Spuneam că neuronii din stratul de ieşire sunt legaţi de neuronii din stratul intermediar şi că ieşirea produsă este doar o sumă ponderată a intrărilor primite de la stratul intermediar. Avem astfel:

$$y = \sum_i w_i h_i \tag{5.4}$$

unde h_i este ieşirea neuronului i din stratul intermediar şi w_i este ponderea legăturii.

Deşi centrii $\mathbf{x}_i^c$ ai funcţiilor radiale au o interpretare foarte clară (neuronul i este activat, adică are ieşirea semnificativă, doar dacă intrarea este *aproape* de centrul neuronului), ponderile w_i nu au o interpretare clară. Putem spune că reţelele neuronale radiale sunt semitransparente.

Se poate arăta că o reţea neuronală radială poate reproduce orice fel de funcţie (este deci un aproximator universal).

Instruirea acestui tip de reţele are două etape:

1. Alegerea sau determinarea centrilor funcţiilor radiale
2. Determinarea ponderilor w_i astfel înc"at o funcţie criteriu să fie minimizată

Alegerea centrilor se poate face distribuindu-i uniform în spaţiul intrărilor. Se poate folosi un algoritm de clasificare pentru a determina aceşti centrii, avantajul fiind faptul că reţeaua va fi mai mică.

Metoda gradientului poate fi folosită în a doua etapă.

S-a observat (vezi [12] sec. 12.3) că acest tip de reţele neuronale converg mai repede dec"at cele din secţiunea precedentă.

Detalii suplimentare legate de reţelele neuronale radiale se pot găsi în [12] sec. 12.3 şi [8] anx. A.4.

Capitolul 6

Maşini bazate pe clusteri

Capitolele anterioare au ca principal scop pregătirea terenului pentru algoritmii ce vor fi prezentaţi în acest capitol care este cel mai important al lucrării. ‘Intreg formalismul şi algoritmii ce vor fi prezentaţi aici sunt rodul muncii de cercetare a autorului pe parcursul a mai mulţi ani (1995–1998) şi, deşi aparent abordarea de faţă seamănă cu o seamă de alte abordări, în literatura de specialitate nimeni nu a pus problema în chear aceşti termeni.

Ultimele două capitole, consacrate unor alte abordări ale problemei aproximării funcţiilor, au pus în evidenţă deficienţe majore ale metodelor prezentate, deficienţe care îngrădesc sfera de aplicabilitate a acestora la probleme concrete.

Scopul acestui capitol este de a prezenta o serie de abordări înrudite ale problemei aproximării funcţiilor care au o seamă de avantaje faţă de abordările actuale. Primul şi cel mai mare avantaj este faptul că algoritmii prezentaţi (at”at cei de învăţare c”at şi cei de utilizare a modelelor construite) sunt mult mai fezabili din punct de vedere computaţional dec”at maşinile fuzzy şi reţelele neuronale.

În principal două tipuri de maşini vor fi definite, maşina quasiconstantă şi maşina quasiliniară. Deşi în multe privinţe diferită, această abordare are certe similitudini at”at cu maşinile fuzzy c”at şi cu reţelele neuronale radiale. Datorită acestor similitudini se poate spune că maşinile prezentate *unifică* cele două formalisme.

Iniţial o încercare de a găsi o metodă eficientă de a antrena maşinile fuzzy, cercetările întreprinse de autor au condus la ideea că introducerea maşinilor mai sus menţionate ar mari mult eficienţa. Idei incipiente ale acestui capitol au fost prezentate de autor în [7].

De asemenea ideile din acest capitol adaugă la lista de utilizări a *Teoriei clusterizării* aproximarea funcţiilor, utilizare ce p”ană acum a fost doar parţială, doar mici porţiuni din algoritmii utilizaţi apelînd la această teorie. Învăţarea maşinilor propuse se bazează în exclusivitate pe *Teoria clusterizării* utilizînd întreaga sa putere şi frumuseţe.

Spre deosebire de celelalte abordările menţionate în ultimele două capitole, abordări în care mai înt"ai s-a ales structura şi mai apoi s-a găsit o metodă de antrenare, în abordarea noastră se înt"amplă chear pe dos, inferenţa fiind adaptată după modul în care se face învăţarea.

6.1 Problema n:1 dimensională

Din motive ce vor fi evidente la prezentarea algoritmilor, construirea de maşini cu mai multe ieşiri este puţin mai problematică şi din acest motiv vom aborda mai înt"ai o varianta monoieşire. Acest lucru justifică numele acestei secţiuni.

Studiul complet al problemei n:1 dimensionale este justificat şi de faptul că algoritmii pentru această variantă mai simplificată sunt puţin mai rapizi şi deci utilizarea lor este recomandată acolo unde nu este necesară varianta m:n dimensională (lucru dictat de aplicaţia practică concretă).

Prezentarea detaliată a noţiunilor de *Teoria clusterizării* (cap. 3) s-a făcut tocmai pentru că aici vom utiliza intens algoritmii prezentaţi acolo.

6.1.1 Maşina quasiconstantă n:1 dimensională

Punctul de plecare în prezentarea maşini quasiconstantă este maşina fuzzy Mamdani (sec. 4.1). Dădeam acolo o interpretare regulilor, care erau privite ca şi modalităţi de comportare în situaţii tipice, şi procesului de inferenţă, care deduce comportarea în situaţii atipice (normale) dată fiind asemănarea cu situaţiile tipice.

Să aruncăm o privire asupra algoritmului de clusterizare fuzzy folosind prototipuri sferice (sec. 3.3.1). Acest algoritm poate servi la determinarea situaţiilor tipice de care pomeneam mai sus.

Considerăm spaţiul intrare-ieşire care are $n+1$ dimensiuni (dacă avem n intrări). 'In acest spaţiu aplicăm algoritmul 4 care ne va furniza un număr p de prototipuri corespunzători celor p nori pe care îi detectăm în datele de antrenament furnizate.

Raţiunea pentru care facem acest lucru este dictată de scopul nostru, acela de a putea scrie reguli de comportare în situaţii tipice. Prototipurile detectate se vor afla doar în zonele în care se află date de învăţare, deci se va furniza informaţie în special pentru zonele care interesează (zonele în care s-au făcut măsurători).

Informaţia obţinută în procesul de clusterizare va fi folosită pentru construirea maşini quasiconstante în modul următor: se proiectează prototipurile $\mathbf{L}_i$ identificate pe spaţiul variabilelor de intrare (se obţine vectorul $\mathbf{L}'_i$) şi separat pe axa variabilei de ieşire (se obţine valoarea $y_i = L_{n+1}$); maşina quasiconstantă va consta din reguli de forma:

$$R_i : \text{If } \mathbf{x} \text{ is near } \mathbf{L}'_i \text{ then } y \text{ is } y_i \tag{6.1}$$

unde cu $\mathbf{x}$ s-a notat vectorul de intrare şi cu y ieşirea.

Teoria fuzzy a clusterizării este folosită şi nu cea clasică deoarece nu există grupări clare în datele de antrenament deoarece datele de intrare sunt relativ uniform distribuite sau în orice caz nu sunt grupate în clase distincte.

Mecanismul de inferenţă, în principiu, va calcula valoarea y a ieşirii ca o combinaţie a valorilor y_i, ponderea fiecăreia fiind determinată de gradul de asemănare a intrării cu $\mathbf{L}'_i$. Astfel:

$$y = \sum_{i=1}^{p} w_i y_i \tag{6.2}$$

unde w_i este ponderea fiecărei reguli. Dorim ca inferenţa să aibă proprietatea de combinaţie convexă (lucru care ar asigura faptul că rezultatul nu va fi mai mare dec"at cea mai mare valoare y_i şi nici mai mic dec"at cea mai mică valoare y_i, presupunere c"at se poate de naturală) ceea ce induce următoarea condiţie de normare pentru ponderile w_i:

$$\sum_{i=1}^{p} w_i = 1 \tag{6.3}$$

Ponderile w_i pot fi privite ca şi funcţii de intrare şi prototipuri ale consecvenţilor

$$w_i = f(\mathbf{x}, \mathbf{L}'_1, ..., \mathbf{L}'_p) \tag{6.4}$$

O altă condiţie c"at se poate de naturală dată fiind interpretarea pe care vrem să o impunem este următoarea:

$$w_i(\mathbf{x} = \mathbf{L}'_j, \mathbf{L}'_1, ..., \mathbf{L}'_p) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{6.5}$$

sau exprimat în cuvinte, c"and intrarea este chear o situaţie tipică ieşirea trebuie să fie chear comportarea tipică corespunzătoare. Condiţia de mai sus trebuie de fapt întărită în sensul că egalităţile să fie înlocuite cu *tinde la*.

Mai sus am impus condiţii ca să avem o comportare bună în jurul prototipurilor doar. Dorim o astfel de comportare şi în spaţiile dintre prototipuri. Astfel comportarea trebuie să fie o *interpolare* a prototipurilor cele mai apropiate.

Putem deriva expresii pentru funcţiile w_i din chear mecanismul de învăţare. După cum am arătat în capitolul 3 în procesul de clusterizare se urmăreşte minimizarea unei funcţii criteriu J. Putem privi sintagma *is near* din expresia regulilor maşini quasiconstante ca fiind apartenenţă la o mulţime fuzzy, astfel ponderile w_i sunt gradele de apartenenţă la aceste mulţimi. Condiţia de normare a ponderilor este în acest caz condiţia ca aceste mulţimi fuzzy să formeze o partiţie a spaţiului de intrare. Funcţiile de apartenenţă ale acestor mulţimi fuzzy ar putea fi generate de prototipuri şi distanţa euclidiană în maniera în care funcţiile de apartenenţă la norii formaţi în procesul de clusterizare sunt generate (vezi ec. 3.49):

$$w_i = \frac{1}{\displaystyle\sum_{k=1}^{n} \frac{d^2(\mathbf{x}, \mathbf{L}'_i)}{d^2(\mathbf{x}, \mathbf{L}'_k)}} \tag{6.6}$$

O astfel de definiţie ar respecta condiţia de normare 6.3 c"at şi condiţia 6.5 dar din păcate se comportă destul de prost printre prototipuri. Din experimente rezultă că acest lucru se datorează faptului că prototipurile îndepărtate *poluează* rezultatul, ponderea lor fiind prea mare. Pentru a evita o astfel de situaţie se pot aborda mai multe metode:

1. să se considere doar cele mai apropiate N prototipuri, valorile pentru N fiind de obicei apropiate de 3

2. să se considere doar prototipurile care au distanţa p"ană la vectorul de intrare mai mică dec"at o anumită valoare

3. să se folosească distanţe mai rapid descrescătoare dec"at distanţa euclidiană, de exemplu:

$$d^2(\mathbf{x}, \mathbf{L}) = \frac{1}{e^{\frac{\|\mathbf{x}-\mathbf{L}\|^2}{2\sigma^2}}} \tag{6.7}$$

   expresie care este chear funcţia de activare de la reţele neuronale radiale.

O altă idee ar fi să se aleagă funcţiile w_i astfel înc"at vectorul $(\mathbf{x}, y)$ să minimizeze funcţia criteriu J dar o astfel de ecuaţie nu are soluţie algebrică şi soluţionarea ei numerică ar reduce mult eficienţa algoritmului. De fapt o soluţie rapidă este de preferat uneia exacte, interesînd o aproximare bună şi ieftină din punct de vedere computaţional şi nu neapărat soluţia optimă.

Chestiuni legate de care din algoritmii de inferenţă este mai bun vor fi trata te în capitolul 8 şi acolo se vor trage şi concluziile de rigoare vizavi de avantajele şi dezavantajele acestora.

Un avantaj evident pe care prima metodă de inferenţă o are (cu puţin efort chear şi a doua) este că se poate folosi o metodă ca nu toţi clusterii să fie consideraţi în procesul de inferenţă ci doar c"aţva (un număr apropiat de N). Acest lucru se poate realiza făcînd o precompilare a mulţimii de prototipuri în modul următor:

1. se împarte spaţiul de intrare în M suprafeţe egale astfel înc"at determinarea suprafeţei căreia îi aparţine un anumit vector de intrare să se facă în timp constant. Dacă se suprapune o grilă peste spaţiu de intrare se obţine o familie de suprafeţe cu o astfel de proprietate

2. se determină pentru fiecare suprafaţa o familie de prototipuri care ar putea fi candidaţi ca făcînd parte dintre cei N cei mai apropiaţi

3. la inferenţă se determină suprafaţa căreia îi aparţine intrarea şi se folosesc la inferenţă doar prototipurile din lista asociată acesteia

Inferenţa are astfel şansa să devină o problemă constantă care să nu depindă de numărul de prototipuri, fiind astfel extrem de eficientă.

Comparînd această maşină cu maşinile fuzzy Mamdani se poate observa că inferenţa (chear şi fără aplicarea metodei mai sus menţionate) este mult mai rapidă presupunînd doar calcularea a p distanţe. Vizavi de reţelele neuronale radiale inferenţa este cel puţin mai ineficientă dacă se foloseşte metoda a treia de inferenţă dar mult mai eficientă dacă se folosesc celelalte două deoarece nu este necesară calcularea unei funcţii neliniare relativ costisitoare ca exponent.

După cum se poate observa maşina quasiconstantă seamănă at"at cu maşina fuzzy Mamdani c"at şi cu reţelele neuronale radiale. Diferenţa esenţială faţă de primele este că nu se consideră funcţii de apartenenţă a fiecărei variabile de intrare ci doar a vectorului de intrare în totalitatea sa şi bine"anţeles nu se consideră nici un fel de funcţie de apartenenţă pentru variabila de ieşire. Faţă de reţelele neuronale radiale diferenţa constă în faptul că ponderile w_i care apăreau în componenţa reţelei primesc aici o semnificaţie clară ca fiind comportări în situaţiile tipice.

Maşina propusă este astfel complet transparentă, regulile ei av"and o interpretare foarte clară. Acesta este un avantaj serios asupra reţelelor neuronale, chear a celor radiale care sunt parţial transparente.

După cum se poate observa din algoritmul de antrenare a maşini nu există ceva asemănător ratei de învăţare de la reţele neuronale şi metoda deşi iterativă are şanse să fie foarte rapidă deoarece constă în minimizări exacte faţă de prototipuri şi funcţii de apartenenţă. Nu se pune problema opririi procesului de învăţare prematur deoarece nu se poate învăţa prea mult.

La terminarea procesului de antrenare se observă că prototipurile sunt un fel de medii aritmetice a punctelor din jur deci maşinile generate sunt foarte robuste, c"ateva date eronate au un efect minor asupra rezultatului final. Din acelaşi motiv posibilităţile de generalizare ale maşini sunt foarte mari.

În procesul de clusterizare nu foloseşte la nimic faptul că una dintre variabile este de intrare şi celelalte de ieşire deci dacă în datele de test am permuta toate valorile unei variabile de intrare cu variabila de ieşire clusterizarea nu ar decurge altfel. ‘Inseamnă că maşinile quasiconstante au proprietatea că inversele faţă de o variabilă de intrare se obţin în mod trivial, permutînd variabila de cu prototipurile ei din antecedent cu variabila de ieşire din consecvent. O astfel de proprietate se poate dovedi extrem de utilă în anumite situaţii. După cunoştinţele autorului nu se cunoaşte nici o metodă asemănătoare pentru reţele neuronale feedforward.

Metoda de antrenare este una bloc, astfel ordinea în care se prezintă datele de antrenament nu are nici o importanţă. După cum se poate observa şi din algoritm o epocă de antrenament are şanse să fie mult mai rapidă dec"at la reţele neuronale deoarece operaţiile efectuate sunt doar puţin mai multe dec"at cele implicate în utilizarea maşini după antrenare. Nu este nevoie de ceva asemănător unui pas forward urmat de unul backward ca în cazul antrenării reţelelor neuronale feedforward.

Din moment ce nu există nici o metodă întradevăr performantă pentru antrenarea maşinilor fuzzy Mamdani am considerat că nu este necesară o comparaţie detaliată între acestea şi maşinile quasiconstante.

Date mai concrete legate de performanţele şi avantajele maşinilor quasiconstante vor fi prezentate în capitolul 8.

Totuşi o problemă răm"ane deschisă: cum se alege numărul de clusteri în procesul de antrenare. Această problemă va fi tratată în ultima secţiune a acestui capitol.

6.1.2 Maşina quasiliniară n:1 dimensională

Ca şi în secţiunea precedentă plecăm tot de la o maşină fuzzy şi anume de la ce Sugeno (vezi sec. 4.2). Din moment ce abordarea acestei situaţii este asemănătoare cu precedenta vom indica doar deosebirile.

Ne propunem deci să construim o maşină asemănătoare cu maşinile fuzzy Sugeno în spiritul secţiunii anterioare. Vom numi această maşină *maşina quasiliniară.*

Plecăm tot de la algoritmul de învăţare. Ideea este de a căuta în mulţimea datelor grupări liniare în loc de cele sferice, grupări care să pună în evidenţă caracterul linear pe porţiuni a setului de date. Pentru aceasta utilizăm algoritmi de clasificare cu prototipuri elipsoizi turtiţi, una din axe fiind mult mai mică dec"at celelalte.

Considerăm deci spaţiul datelor de intrare-ieşire $n+1$ dimensional. Pentru ca prototipurile clusterilor să aibă forma mai sus menţionată trebuie să folosim o combinaţie convexă între două funcţii criteriu, cea sferică şi cea liniară (vezi sec. 3.3.3). Partea sferică asigură localizarea clusterilor şi partea liniară asigură caracterul linear pe care dorim să-l utilizăm. Partea liniară este de fapt un hiperplan n dimensional.

Distanţa folosită în procesul de clusterizare este:

$$d^2 = \varepsilon d_l^2 + (1-\varepsilon) d_s^2 \tag{6.8}$$

unde d_s este distanţa sferică (euclidiană), d_l este distanţa la hiperplan, ε este elipsoidalitatea, $\varepsilon = 0$ face ca distanţa să fie pur euclidiană.

Ca şi în cazul precedent maşina quasiliniară va fi formată din p reguli, fiecare corespunzînd unui cluster identificat în setul de date. Prototipurile de această dată sunt formate dintr-un punct $\mathbf{v}_i$ şi un set de n vectori generatori ai planului care trece prin acest punct. Se poate trece la o reprezentare mai convenabilă, un punct şi un vector perpendicular pe plan. Acest vector perpendicular pe plan se poate obţine prin două metode:

1. prin rezolvarea unui sistem cu n ecuaţii şi n necunoscute aşa cum este sugerat în [7]

2. din informaţia disponibilă în procesul de clasificare. Astfel vectorii directori sunt vectorii proprii corespunzători celor mai mari n valori proprii ai matricii de împrăştiere S (vezi alg. 7). Dar conform unei teoreme celebre (vezi [17] cap. 2 teorema 4.2) vectorii proprii ai unei matrici simetrice (cum este S) sunt ortogonali. În particular vectorul propriu corespunzător celei mai mici valori proprii este perpendicular pe

toţi ceilalţi şi deci şi pe hiperplanul generat de punctul $\mathbf{v}_i$ şi aceştia. Notăm acest vector cu $\mathbf{u}_i$.

Conform [6] ecuaţia planului determinat de un punct şi un vector perpendicular pe plan este:

$$u_{i1}(x_1 - v_{i1}) + ... + u_{in}(x_n - v_{in}) + u_{in+1}(y - vin + 1) = 0 \tag{6.9}$$

ceea ce se poate rescrie într-o formă convenabilă scopului urmărit:

$$y = c_{i0} + x_1 c_{i1} + ... + x_n c_{in} \tag{6.10}$$

$$c_{i0} = v_{in+1},\ c_{i1} = -\frac{u_{i1}}{u_{in+1}}, ..., c_{in} = -\frac{u_{in}}{u_{in+1}} \tag{6.11}$$

Acestea fiind spuse regulile maşinii sunt de forma:

$$R_i : \text{If } \mathbf{x} \text{ is near } \mathbf{v}_i' \text{ then } y_i = c_{i0} + x_1 c_{i1} + ... + x_n c_{in} \tag{6.12}$$

unde $\mathbf{v}_i'$ e proiecţia vectorului $vecv_i$ pe subspaţiul variabilelor de intrare.

În implementarea acestui algoritm am folosit a doua metodă fiind mult mai economică dec"at prima. De fapt această a doua metodă are un alt avantaj care poate fi exploatat şi anume faptul că nu toate valorile şi vectorii proprii ai matricii S trebuie determinaţi ci doar ce mai mică valoare şi vectorul corespunzător. Din moment ce există o metodă de a le determina doar pe acestea şi este mult mai puţin costisitoare dec"at determinarea tuturor valorilor şi vectorilor proprii există posibilitatea ca acest lucru să fie exploatat în implementare.

Ca mecanism de inferenţă poate fi folosit cel pentru maşini quasiconstante, întreaga discuţie din secţiunea anterioară răm"anînd valabilă. Totuşi din experimentele efectuate s-a observat că prototipurile care conţin plane cu panta mare au tendinţa să influenţeze rezultatul pe o porţiune mai mare dec"at cele cu panta mică. Putem face o corecţie în acest sens considerînd că distanţa dintre intrare şi proiecţia prototipului pe subspaţiul intrărilor să nu fie una euclidiană în acest spaţiu ci să fie de fapt distanţa dintre punctul de pe planul prototipului a cărui proiecţie este intrarea actuală şi centrul de greutate a prototipului. Coordonata y a punctului de care vorbeam se obţine din ecuaţia hiperplanului. Avem astfel:

$$d^2(\mathbf{x}, \mathbf{L}'_i) = d^2(\mathbf{x}, \mathbf{v}'_i) + (y - v_{in+1})^2 \tag{6.13}$$

$$u_{i1}(x_1 - v_{i1}) + ... + u_{in}(x_n - v_{in}) + u_{in+1}(y - v_{in+1}) = 0 \tag{6.14}$$

$$y - v_{in+1} = \frac{u_{i1}(x_1 - v_{i1}) + ... + u_{in}(x_n - v_{in})}{u_{in+1}} \tag{6.15}$$

$$y - v_{in+1} = c_{i1}(x_1 - v_{i1}) + ... + c_{in}(x_n - v_{in}) \tag{6.16}$$

$$\begin{aligned} d^2(\mathbf{x}, \mathbf{L}'_i) = {} & [c_{i1}(x_1 - v_{i1}) + ... + c_{in}(x_n - v_{in})]^2 + \\ & + (x_1 - v_{i1})^2 + ... + (x_n - v_{in})^2 \end{aligned} \tag{6.17}$$

unde variabilele au semnificaţia folosită pe parcursul secţiunii trecute şi a acesteia.

Această maşină are avantajul că într-un spaţiu relativ mic concentrează o cantitate mare de informaţie. Modelul este de fapt o compunere de modele liniare. Un subiect important legat de acest fapt (care nu este tratat în această lucrare) este acela al proprietăţilor părţilor liniare care sunt preluate de modelul global.

Comparativ cu metodele semieuristice de identificare a maşinilor fuzzy Sugeno, metoda indicată aici este mai performantă, av"and o justificare teoretică solidă.

Pe l"angă vectorii proprii ai matricilor S din care se calculează ecuaţiile hiperplanelor, valorile proprii pot servi informaţii foarte importante a căror utilizare poate aduce importante avantaje. Ne referim în primul r"and la faptul că valorile proprii sunt de fapt măsuri ale împrăştierii norilor în direcţiile vectorilor proprii. Deci cea mai mică valoare proprie poate fi folosită ca măsură a erorii cu care se aproximează local funcţia neliniară de identificat cu un hiperplan. Acest lucru se poate dovedi extrem de util în majoritatea aplicaţiilor. De asemenea el poate fi utilizat în determinarea dimensiunii maşini (a numărului de reguli sau clusteri). Această oportunitate nu este totuşi exploatată în implementarea curentă studiul ei răm"anînd un subiect de studiat în viitor.

După cum vom arăta în capitolul 8 această metodă are şansa să poată aproxima o funcţie neliniară cu o foarte mare precizie.

O observaţie importantă care se referă la toate modalităţile de aproximare a funcţiilor printre care se numără şi reţelele neuronale, maşinile fuzzy şi maşinile bazate pe clusteri este aceea că pot fi privite ca modalităţi de reprezentare a cunoştinţelor legate de funcţii neliniare, cunoştinţe care nu prea pot fi altfel reprezentate altfel. Din acest punct de vedere maşinile quasiliniare sunt cea mai compactă şi mai eficientă reprezentare.

Metodele care se bazează pe minimizarea erorii (dreapta de regresie, metoda gradientului, etc.) nu ţin seama de zgomotul din datele de intrare ci doar de cel din datele de ieşire. Acest lucru nu este valabil şi pentru maşinile quasiliniare deoarece în timpul antrenării nu se face distincţie între intrare şi ieşire, toate fiind tratate în mod egal. Considerăm din acest motiv că rezultatele produse cu această metodă chear şi pentru funcţii liniare (caz în care o singură regulă e suficientă) sunt mai de încredere comparativ cu rezultatele produse de celelalte metode mai sus menţionate.

Ca şi în cazul maşinilor quasiconstante, calculul inversei maşini quasiliniare faţă de o variabila'de intrare şi cea de ieşire este trivial, reducîndu-se doar la rezolvarea ecuaţiei 6.9 faţă de noua variabilă de ieşire şi la proiecţia vectorului $\mathbf{v}_i$ pe noul subspaţiu de intrare.

Vom spune mai multe despre avantajele maşinilor quasiliniare în capitolul 8.

6.2 Problema m:n dimensională

Aşa cum am discutat şi la maşini fuzzy Mamdani, maşina cu m variabile de ieşire poate fi considerată ca fiind echivalentă cu m maşini cu o singură ieşire. Acest lucru revine la a determina m maşini în procesul de învăţare şi la a utiliza m maşini (a face m inferenţei separate) mai apoi. Din păcate o astfel de abordare, pe l"angă faptul că este costisitoare din punct de vedere calculaţional, nu exploatează legăturile dintre variabile.

O altă metodă de soluţionare a acestei probleme consideră variabila de ieşire o variabilă vectorială, toate ecuaţiile în care aceasta intră fiind vectoriale. 'In acest fel s-ar face o distincţie prea dură între variabilele de intrare şi cele de ieşire pierzîndu-se unul dintre avantajele abordării propuse.

Varianta pe care o adoptăm în continuare este aceea de a considera variabilele de ieşire ca variabile simple ca şi cele de intrare. Partea de antecedent a regulilor (calculul ponderilor) se va face o singură dată aplicîndu-se apoi pentru fiecare variabilă de ieşire ecuaţiile din cazul n:1 dimensional.

Din moment ce există unele diferenţe în modul în care se poate face antrenarea celor două tipuri de maşini propuse în cazul m:n dimensional vom trata separat fiecare dintre ele.

6.2.1 Maşina quasiconstantă m:n dimensională

În cazul maşini quasiconstante problema învăţării răm"ane în esenţă aceiaşi. Se utilizează un algoritm de clusterizare fuzzy pe datele de antrenament din spaţiul intrare-ieşire $n+m$ dimensional. Pentru a construi maşina se proiectează prototipurile rezultate pe spaţiul intrărilor şi separat pe spaţiul ieşirilor şi se scriu reguli de forma:

$$R_i : \text{If } \mathbf{x} \text{ is near } \mathbf{L}_i' \text{ then } \mathbf{y} \text{ is } \mathbf{y}_i \tag{6.18}$$

unde $\mathbf{L}_i'$ este proiecţia prototipului $\mathbf{L}_i$ pe spaţiul de intrare şi $\mathbf{y}_i$ e proiecţia sa pe spaţiul de ieşire.

Formula de inferenţă este vectorială de această dată, putîndu-se descompune în m formule scalare:

$$\mathbf{y} = \sum_{i=1}^{p} w_i \mathbf{y}_i \tag{6.19}$$

Ponderile w_i au aceiaşi semnificaţie şi se obţin în modul în care s-a discutat la cazul n:1 dimensional.

Distanţa care se foloseşte în procesul de clusterizare este natural de definit ca:

$$d^2((\mathbf{x},\mathbf{y}),\mathbf{L}) = (x_1 - L_1)^2 + ... + (x_n - L_n)^2 + (y_1 - L_{n+1})^2 + ... + (y_m - L_{n+m})^2 \tag{6.20}$$

Toate proprietăţile pe care le-am enumerat în cazul n:1 dimensional răm"an valabile şi aici.

6.2.2 Maşina quasiliniară m:n dimensională

Varianta m:n dimensională a maşinii quasiliniară este puţin mai dificil de obţinut deoarece considerarea a m ieşiri înseamnă găsirea a m relaţii liniare în cadrul unui cluster (m hiperplane), ceea ce revine la construirea a m matrici de împrăştiere.

Putem privi problema şi într-un alt mod. Dorim ă găsim m clusteri elipsoidali turtiţi care însă să aibă aceiaşi localizare, fapt care ne-ar ajută să folosim aceleaşi ponderi în procesul de inferenţă pentru toate variabilele de ieşire.

Distanţa pe care o vom folosi în procesul de clusterizare este:

$$d^2 = \varepsilon \frac{\sum_{j=1}^{m} d_{lj}^2}{m} + (1-\varepsilon) d_s^2 \tag{6.21}$$

unde se observă că ţinem seama de toate distanţele la hiperplanele unui prototip.

Matricile de împrăştiere vor fi matrici pătratice de dimensiune $n+1$, ele vor fi obţinute luînd variabilele de intrare şi c"ate o variabilă de ieşire şi aplicînd formulele de calcul a matricilor de împrăştiere.

Această modalitate de a realiza clasificarea nu a mai fost menţionată în literatura de specialitate şi studiul ei nu este complet, neexistînd teoreme de convergenţă ca cele pentru cazurile considerate în capitolul 3.

O altă observaţie importantă este legată de faptul că nu se mai poate folosi corectura la procesul de inferenţă care a fost menţionată la cazul n:1 dimensional.

Această generalizare deşi importantă este doar teoretică în acest moment, studii ulterioare vor fi consacrate analizei acesteia.

6.3 Determinarea dimensiunii optime a maşinilor

Problema determinării structurii optime a maşinii, deşi complicată în cazul reţelelor neuronale, poate fi tratată cu destul de multă uşurinţă în cazul maşinilor bazate pe clusteri.

Dimensiunea maşini se alege în funcţie de mai multe criterii:

1. precizia dorită a rezultatului (se alege mai mare sau egală cu zgomotul din datele de antrenament, o valoare mai mică nu este indicată deoarece generalizarea maşini scade drastic datorită faptului că sub pragul de zgomot se vor învăţa şi date înt"amplătoare)

2. timpul de antrenament şi cel de calcul (se pot estima cu mare uşurinţă dacă se cunosc date de test pentru o maşină de acelaşi tip antrenată pe acelaşi suport hardware deoarece timpul de antrenament este proporţional cu numărul datelor de antrenament şi cu numărul de clusteri)

3. resursele de memorie disponibile (uşor de determinat, este cea mai slabă constr"angere deoarece spaţiul de memorare necesar este mic)

Dacă se impune doar precizia rezultatului se pot utiliza şi algoritmi de clusterizare cu determinare dinamică a numărului de clusteri (vezi [9] cap. 4). Aceşti algoritmi, în cazul maşinilor quasiliniare, ar putea utiliza cea mai mică valoare proprie a matricii de împrăştiere ca măsură a dispersiei clusterilor şi pe baza acesteia să decidă dacă local este nevoie de o structură de clusteri mai rafinată.

Punerea la punct a unor astfel de metode va face subiectul unui studiu ulterior.

Exemple practice şi detalii suplimentare legate de alegerea dimensiuni optime a maşinilor vom prezenta în capitolul 8.

6.4 Posibilităţi de paralelizare a algoritmilor

Analizînd at”at algoritmul 5 c”at şi algoritmul 7 precum şi implementarea acestora în pachetul `clus` se pot observa următoarele vizavi de posibilităţile de paralelizare a acestora.

1. o mare parte din calculele care se execută la fiecare pas at”at în procesul de învăţare c”at şi în cel de utilizare (şi anume cele care sunt executate de obiectele de tip cluster, vezi capitolul 7) pot fi executate în paralel, neexistînd vreo interacţiune cu celelalte

2. celelalte calcule presupun sau distribuirea unor valori (a intrării la fiecare cluster) sau efectuarea unor sume, existînd pentru ambele implementări paralele eficiente (în timp logaritmic)

3. o altă sursă de paralelism sunt paşi elementari (în care se prezintă o pereche intrare-ieşire) ai unei epoci de învăţare deoarece pentru aceste maşini nu are importanţă în ce ordine se prezintă datele de antrenament. Pentru fiecare dintre clusteri s-ar putea aloca mai multe procese care să efectueze calculele necesare în paralel lucrînd pe seturi distincte de date, la sf”arşitul fiecărei epoci făcîndu-se doar centralizarea acestora (tot nişte sume)

Cele spuse mai sus ne îndreptăţesc să spunem că o implementare hardware a acestor maşini (dar mai ales a celei quasiconstante) ar fi destul de uşor de realizat.

Partea III

Implementare şi testare

Capitolele următoare sunt dedicate implementării şi testării ideilor exprimate pe parcursul cărţii referitoare la maşinile quasiconstante şi quasiliniare.

În partea de implementare este prezentat programul `clus` a cărui listing se poate găsi în anexă iar în partea de testare se utilizează acest program pentru a evalua din punct de vedere practic algoritmii prezentaţi şi pentru a confirma idei exprimate pe parcursul lucrării

Capitolul 7

Implementare

În acest capitol vom prezenta structura generală, detalii de implementare şi instrucţiuni de utilizare a programului `clus`.

Ca şi limbaj de programare a fost ales *Limbajul C++* din următoarele motive:

1. problema aşa cum va reieşi din secţiunile următoare poate fi modelată foarte bine de un limbaj obiectual, utilizarea unuia imperativ îngreunînd mult claritatea şi chear calitatea implementării

2. este nevoie de un limbaj de programare foarte rapid deoarece algoritmii implementaţi sunt mari consumatori de timp de calcul; problema se pretează la optimizări majore în faza de compilare iar compilatoarele de C++ sunt printre puţinele compilatoare care pot genera cod întradevăr rapid

3. compilatoare de C++ sunt disponibile pe mai toate platformele

Limbajul C era un candidat serios dar din cauza primului motiv nu a fost ales. Limbajul Java este mult prea lent pentru o astfel de problemă în care viteza este esenţială şi care este suficient de simplă pentru a nu exista probleme cu alocarea şi dealocarea spaţiului de memorie.

Pe viitor pentru a creşte şi mai mult viteza (de două ori după opinia autorului) se poate translata actuala împlementare în limbajul C, folosind eventual biblioteci speciale pentru procesoare MMX.

Fiind o implementare a unei idei teoretice am considerat că o interfaţă grafică nu este necesară, toate programele furnizate fiind apelate din linia de comandă. Aceasta datorită faptului că numărul de parametri este în general mic (maxim 6) şi că o interfaţă grafică nu ar fi uşurat cu nimic utilizarea.

Acest pachet de programe se ocupîndu-se în principal cu aproximarea funcţiilor este necesară utilizarea unui program pentru reprezentarea grafică (dacă au maxim două variabile de intrare şi una de ieşire). Orice program de reprezentare căruia funcţia de reprezentat i se poate specifica într-un fişier (ca o listă de puncte, c"ate unul pe r"and) poate fi folosit. Autorul acestei lucrări a utilizat (şi de asemenea indică folosirea) programului `gnuplot` sub *XWindows* care este disponibil pe toate platformele UNIX. Toate reprezentările grafice din capitolul 8 sunt realizate cu acest produs.

Reprezentarea grafică a rezultatelor este necesară doar în perioada de acomodare cu produsul precum şi în cea de apreciere a performanţelor, eventualele erori fiind uşor de sesizat într-o astfel de reprezentare.

Deşi foarte rapizi comparativ cu alte abordări similare (vezi capitolele anterioare pentru explicaţii), algoritmii implementaţi sunt costisitori din punct de vedere al timpului de calcul. Acest lucru ne face să recomandăm utilizarea unor calculatoare puternice (cel puţin echivalente cu cele din familia Pentium Intel).

Ca o ultimă observaţie, codul scris este portabil, neconţinînd nici o secvenţă dependentă de vreo maşină sau sistem de operare, fiind eventual necesară modificarea fişierului `Makefile`. O eventuală interfaţă grafică ar fi redus această portabilitate şi nu ar fi adus mare lucru în plus.

7.1 Structura generală a programului

După cum spuneam întreg programul este implementat în limbajul C++. Există un nucleu de clase bine definite şi c"ateva utilitare care folosesc aceste clase. În această secţiune vom prezenta structura de clase, arătînd scopul fiecăreia şi eventuale comentarii interesante legate de acestea fără a prezenta definiţia exactă sau detalii concrete de implementare, lucruri care pot fi uşor identificate în codul sursă listat la sf"arşitul lucrării.

Aproape tot codul este scris în manieră obiectuală prin definiri de clase. Excepţie fac două funcţii: `SolveSystem` şi `Jacobi`, prima dintre ele rezolvă un sistem de n ecuaţii cu n necunoscute iar a doua determină vectori şi valorile proprii ale unei matrici reale, simetrice şi pozitiv definite utilizînd metoda lui Jacobi. Detalii suplimentare legate de implementarea acestor funcţii vor fi furnizate la sf"arşitul secţiunii

Figura 7.1 este o reprezentare grafică a claselor sub forma unei ierarhi de generalizare-specializare, în partea de sus a ovalelor este specificat numele clasei, în partea de jos fişierele în care se află sursa (declaraţia şi definiţia) respectivei clase. La implementare s-a ţinut seama de următoarele convenţii:

1. într-un fişier header s-a pus definiţia unei singure clase, numele fişierului fiind o prescurtare (nu mai lungă de 8 caractere pentru a nu exista probleme cu sistemele de fişiere care nu suportă nume mai lungi) a numelui clasei

2. funcţiile inline au fost definite în fişierul cu acelaşi nume ca al fişierului header dar cu extensia `.icc`

3. celelalte funcţii au fost definite în fişierul cu acelaşi nume ca al fişierului header dar cu extensia `.cc`

O seamă de comentarii utile se pot găsi chear în codul sursă.

Vom împărţi clasele în categorii pentru o prezentare mai facilă.

7.1.1 Clase pentru modelarea învăţării

Aceste clase implementează efectiv ideile din capitolele precedente referitoare la maşinile quasiconstante şi quasiliniare. După cum se poate observa şi din prezentarea teoretică implementarea acestor maşini se poate face foarte structurat, fiecare dintre ele constînd dintr-un număr de clusteri care interacţionează destul de puţin între ei.

Datorită faptului că interacţiunea între clusteri este la fel în ambele cazuri (quasiconstant şi quasiliniar) nu este necesară implementarea de două clase distincte pentru fiecare tip de maşină. Figura 7.2 este o ilustrare grafică a structurii unei astfel de maşini.

O astfel de structură se datorează posibilităţilor de paralelizare a algoritmilor prezentaţi. Chear şi o implementare hardware destul de simplă este posibilă dar un astfel de implementare nu face subiectul acestei lucrări.

Clusteri

Sub acest nume se ascund obiectele care pot intra în componenţa unei maşini pentru a realiza structura din figura 7.2.

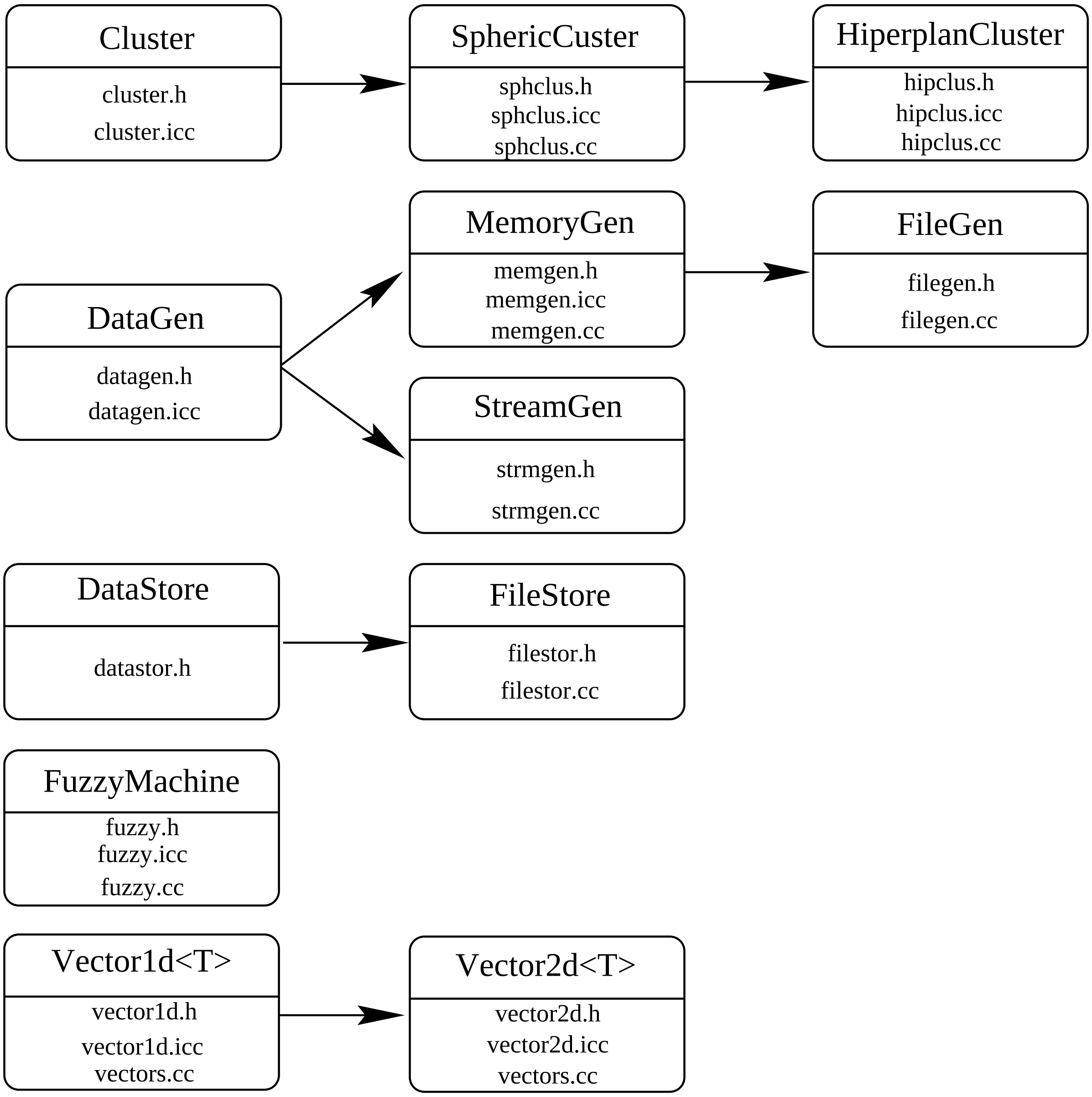


Figura 7.1: Ierarhia de clase. Săgeţile semnifică relaţii de specializare

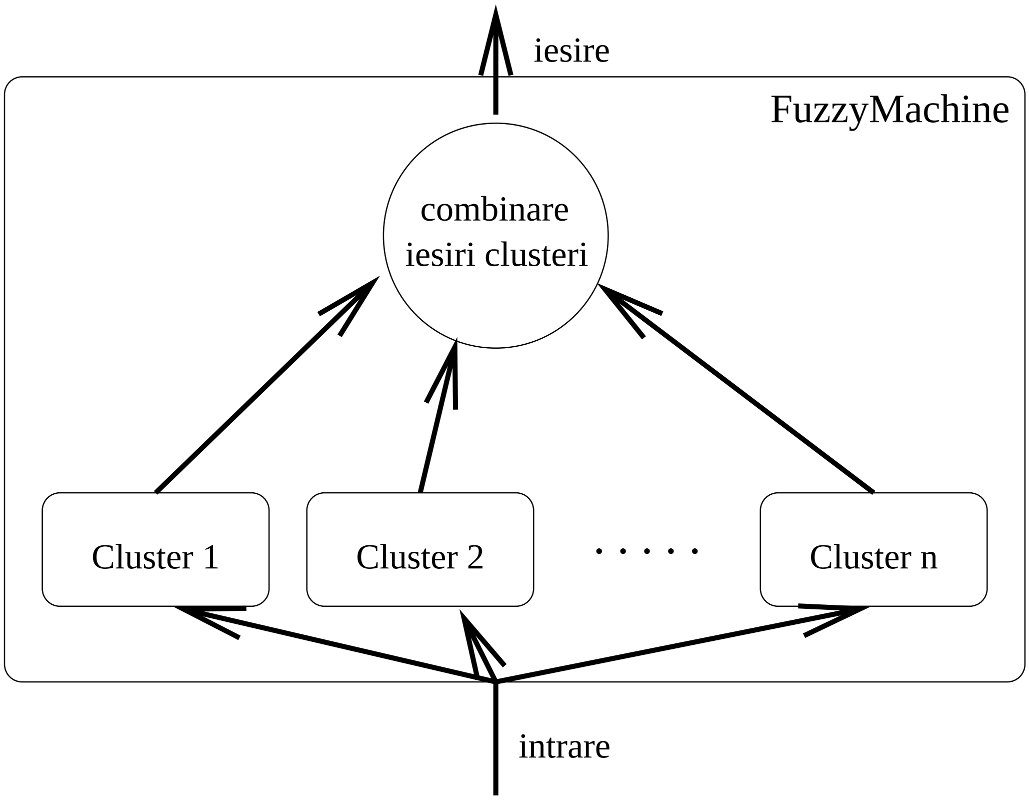


Figura 7.2: Structura generală a maşinilor cu clusteri (clasa *FuzzyMachine*).

Clasa Cluster este clasa abstractă care defineşte interfaţa pe care orice obiect de tipul cluster trebuie să o ofere. Această interfaţă este utilizată de obiectele din clasa *FuzzyMachine* pentru a realiza efectiv inferenţa şi utilizarea. Declaraţia ei se află în fişierul `cluster.h` iar metodele ei neabstracte sunt declarate în fişierul `cluster.icc`

Clasa SphericCluster este o specializare concretă a clasei *Cluster* care serveşte la implementarea clusterilor sferici necesari construirii maşini quasiconstante. Codul pentru această clasă este derivat din algoritmul 5 şi se găseşte în fişierele `sphclus.h`, `sphclus.icc` şi `sphclus.cc`.

Clasa HiperplanCluster este o specializare a clasei *SphericCluster* deşi aparent ar trebui să fie o specializare a clasei *Cluster* deoarece majoritatea codului scris poate fi refolosit. Obiectele instanţe ale acestei clase sunt constituienţi ai unui obiect de tip *FuzzyMachine* dacă se doreşte implementarea unei maşini quasiliniare. Codul acestei clase se găseşte în fişierele `hipclus.h`, `hipclus.icc` şi `hipclus.cc`; este derivat din algoritmul 7.

În procesul de învăţare aceşti clusteri au nevoie de o metodă de determinare a vectorilor şi a valorilor proprii. Pentru acest scop se utilizează funcţia *Jacobi*.

Maşini

Maşinile sunt obiecte care sunt capabile să înveţe un set de date de test sau să folosească cele învăţate pentru a prezice valori de funcţii, pentru îndeplinirea acestor scopuri fiind capabile să folosească obiecte instanţe a claselor de intrare-ieşire. În această implementare am construit un singur tip de maşină, maşină care generalizează maşinile quasiconstante şi quasiliniare.

Clasa FuzzyMachine implementează partea comună a algoritmilor 5 şi 7. După cum spuneam figura 7.2 prezintă structura acestei clase, structură care, aşa cum se poate observa, seamănă foarte mult cu o reţea neuronală radială. Deosebirea faţă de aceasta constă doar în modul în care se fac operaţiile în stratul de ieşire. Pe l"angă coordonarea învăţării şi a utilizării, obiectele instanţe ale acestei clase sunt capabile să-şi memoreze şi să-şi refacă structura dintr-un fişier. Codul se găseşte în fişierele `fuzzy.h`, `fuzzy.icc` şi `fuzzy.cc`.

7.1.2 Clase de intrare ieşire

Aceste clase asigură generarea şi memorarea datelor (de exemplu a datelor de antrenament sau a rezultatelor procesului de inferenţă). Sunt clase auxiliare care servesc salvării şi recuperării datelor de pe diferite suporturi de memorare.

Clase de intrare

Se mai numesc generatoare de date şi sunt concepute să preia sau să producă date de pe un anumit suport, eventual efectuînd prelucrări suplimentare cum ar fi normalizările.

Clasa DataGen este clasa abstractă care specifică interfaţa comună tuturor generatoarelor de date. Este declarată în fişierul `datagen.h`

Clasa MemoryGen este o specializare a clasei *DataGen* şi oferă facilităţi de normalizare a datelor (vezi 3.1.3). Datele sunt în întregime depozitate în memorie, utilizarea lor fiind foarte rapidă. Declaraţia acesteia se găseşte în `memgen.h` iar definiţiile metodelor în `memgen.icc` şi `memgen.cc`.

Clasa FileGen particularizează clasa *MemoryGen* pentru cazul în care datele sunt memorate în fişier. Este clasa care este folosită dacă datele sunt folosite în mod repetat, de exemplu la învăţare unde sunt mai multe epoci. Codul se află în fişierele `filegen.h` şi `filegen.cc`.

Clasa StreamGen este o specializare a clasei *DataGen* pentru cazul în care datele sunt memorate în fişier dar (spre deosebire de clasa *FileGen*) datele sunt necesare în ordine şi numai o singură dată. O astfel de situaţia se înt"alneşte la utilizarea maşinilor pe un set de date de test. Codul se află în fişierele `strmgen.h` şi `strmgen.cc`.

Clase de ieşire

Aceste clase asigură memorarea datelor în fişiere astfel înc"at, cu ajutorul claselor mai sus prezentate sau cu alte programe (`gnuplot` în speţă) ele să poată fi folosite. Clasele de ieşire trebuie să fie capabile să preia date şi să le memoreze.

DataStore Această clasă abstractă specifică interfaţa pe care orice clasă de ieşire trebuie să o ofere eventualelor clase client. Structură unei astfel de clase este descrisă în fişierul `datastor.h`.

FileStore Este o implementare concretă de clasă de ieşire care memorează datele într-un fişier într-un format acceptat de `gnuplot` (fiecare r"and conţine un set de date). Declaraţia clasei se găseşte în fişierul `filestor.h` iar definiţia metodelor sale în `filestor.cc`.

7.1.3 Vectori şi matrici

În acest program sunt utilizaţi din prin vectori şi matrici. S-a preferat implementarea lor sub formă de clase template deoarece s-a putut identifica un nucleu operaţional comun tuturor tipurilor generale de astfel de structuri de date (incomplet în general dar suficient scopurilor acestui program).

S-a urmărit în primul r"and scrierea unui cod c"at se poate de rapid şi doar în al doilea r"and scrierea unui cod comod din punct de vedere al utilizatorului. Clasa care implementează vectorii este *Vector1d*. Cea care implementează matricile este *Vector2d* şi moşteneşte clasa *Vector1d*.

Din moment ce nu sunt probleme deosebite în implementarea acestor clase nu vom detalia nici structura lor care se găseşte în fişierele `vector1d.h` şi `vector2d.h`, nici detaliile de implementare (fişierele `vector1d.icc, vector2d.icc` şi `vectors.cc`).

7.1.4 Funcţii definite independent de clase

Aceste funcţii sunt declarate în fişierul header `eigval.h` şi definite în fişierul `eigval.cc`. Aceste funcţii nu au fost ataşate nici unei clase ca metode deoarece am considerat că acest lucru ar fi oarecum forţat.

Funcţia SolveSystem Rezolvă un sistem de ecuaţii (cu numărul de ecuaţii egal cu cel al necunoscutelor şi numai dacă sistemul este compatibil determinat, adică dacă are o soluţie şi dacă este unică) utilizînd eliminarea gaussiană cu pivotare parţială (vezi algoritmul 2.3 din [14]).

Funcţia Jacobi Această funcţie implementează algoritmii 9.10 şi 9.11 expuşi în [14]. Metoda lui Jacobi permite determinarea concomitentă at"at a valorilor proprii c"at şi a vectorilor proprii. Această metodă a fost aleasă doar ca primă metodă de determinare a vectorilor şi valorilor proprii, alte metode urmînd să fie folosite dacă se urmăreşte creşterea vitezei maşinilor ca de exemplu algoritmul QZ (vezi [14]).

Deoarece problema concretă nu cere ca toţi vectori proprii şi valorile proprii corespunzătoare să fie determinaţi ci doar cea mai mică valoare proprie şi vectorul asociat se poate utiliza algoritmul 9.9 din [14] pentru a găsi cea mai mică valoare proprie şi o rezolvare de sistem de ecuaţii pentru a determina vectorul corespunzător. Această ultimă metodă s-ar putea (autorul nu dispune de studii de performanţă a acestor algoritmi) să fie foarte eficientă pentru dimensiuni mari (peste 5 probabil).

7.2 Prezentarea principalelor utilitare

Vom prezenta mai jos utilitarele care intră în componenţa pachetului `clus`. Ele au fost implementate utilizînd clasele descrise mai sus. În legătură cu fiecare din acestea vom menţiona utilitatea, modul intern de funcţionare şi utilizarea prin specificarea parametrilor din linia de comandă. Exemple practice de utilizare vor fi indicate în capitolul următor unde pentru fiecare experiment efectuat vom indica şi secvenţa de comenzi folosite.

Pe l"angă programele care ţin de pachetul `clus` vom prezenta pe scurt programul `gnuplot` care este utilizat pentru reprezentări grafice bi şi tri dimensionale.

Toate aceste programe pot fi utilizate ca exemple de folosire a claselor descrise mai sus, putîndu-se uşor integra în orice aplicaţie.

O eventuală aplicaţie ar putea folosi chear aceste utilitare în formă binară apelîndu-le ca să îndeplinească anumite sarcini în maniera în care multe din interfeţele grafice în sistemele UNIX lucrează.

Vom utiliza, şi sugerăm şi eventualilor utilizatori s-o facă, următoarele convenţii de denumire a fişierelor (alegere a extensiei):

1. fişierele care conţin descrieri de funcţii (fie ele de antrenament sau produse de utilitare) au extensia `.dta`

2. fişierele care conţin descrieri de maşini au extensia `.fm`

3. fişierele care conţin secvenţe de animaţie au extensia `.gpl`

4. fişierele care conţin evoluţia erorii medii pătratice în procesul de învăţare au extensia `.hst`

Alte convenţii de denumire a fişierelor în exemplele ce vor fi prezentate sunt:

1. fişierele care conţin datele de antrenament au nume care începe cu `fig`

2. fişierele care conţin prognoze ale valorilor funcţiei învăţate au nume care încep cu `rez`

Pentru fiecare program se va preciza utilitatea şi va fi prezentată lista de opţiuni posibile, urmînd ca exemple de utilizare să fie prezentate în capitolul 8.

7.2.1 Programul `clus`

Acest program are ca scop construirea de modele quasiconstante sau quasiliniare pentru un set de date de antrenament. Sursa sa este fişierul `clus.cc`.

Programul afişiază pe ieşirea standard cum evoluează procesul de învăţare (valoarea funcţiei J la sf"arşitul fiecărei epoci).

Funcţionează doar din linie de comandă şi are 10 opţiuni. În continuare ne vom ocupa cu explicarea acestor opţiuni.

1. **-h** afişează un scurt mesaj informativ referitor la modul de utilizare

2. **-g numefişier** specifică fişierul cu date de antrenament, fişier în formatul celor acceptate de `gnuplot`. Acest parametru este obligatoriu.

3. **-f numefişier** specifică fişierul care conţine o descriere iniţială a maşinii care urmează a fi rafinată prin învăţare. Este utilizată dacă se doreşte de exemplu testarea unor modalităţi avantajoase de iniţializare a maşinii. Acest parametru este opţional dar dacă nu este prezent o maşină aleatoare iniţială va fi generată.

4. **-n numărintreg** specifică numărul de clusteri pe care maşina îl va avea. Acest parametru este opţional, dacă nu apare parametrul **-f** trebuie să fie neapărat prezent, şi invers.

5. **-S** specifică faptul că se vor utiliza clusteri de formă sferică (adică maşina generată este de tipul quasiconstant). Apare obligatoriu în conjuncţie cu opţiunea **-n**.

6. **-H** specifică faptul că se vor utiliza clusteri de formă elipsoidală (adică maşina generată este de tipul quasiliniar). Apare obligatoriu în conjuncţie cu opţiunea **-n**, dar nu c"and s-a folosit opţiunea **-S**, şi viceversa.

7. **-t numărreal** specifică toleranţa procesului de învăţare. C"and variaţia de la o epocă la alta a criteriului calităţii învăţării (funcţia J normată la numărul de date de test şi de clusteri) scade sub acest prag învăţarea se opreşte. Dacă acest parametru nu este prezent se consideră valoarea acestuia 10^-3.

8. **-i numărintreg** specifică numărul maxim de epoci ale procesului de învăţare. Dacă acest parametru nu este prezent se consideră că valoarea sa este 30.

9. **-F numefişier** specifică numele fişierului în care se va memora maşina generată.

10. **-p pattern** specifică un pattern de nume pentru maşinile generate, c"ate una pentru fiecare epocă, reprezentînd stadiile în care se găsea maşina după fiecare iteraţie. Patternul trebuie să conţină secvenţa `%d` pentru a fi valid. Dacă este prezentă această opţiune, precedenta poate lipsi.

7.2.2 Programul `infer`

Acest program produce aproximaţii ale funcţiilor utilizînd maşini quasiconstante sau quasiliniare produse cu programul `clus`. Sursa acestuia este fişierul `infer.cc`.

Pentru a fi utilizat este necesară specificarea unora dintre următoarele opţiuni:

1.

2. **-h** afişează un scurt mesaj informativ referitor la modul de utilizare

3. **-g numefişier** specifică fişierul cu date de intrare în care se va face aproximarea funcţiei. Acest parametru este obligatoriu.

4. **-f numefişier** specifică fişierul care conţine descrierea maşini utilizate. Acest nume poate fi un patern de nume de genul celor folosite de programul `clus`, caz în care pentru fiecare maşină a cărei nume se obţine prin înlocuirea caracterelor `%d` cu un număr întreg se va produce un fişier de ieşire. Argumentul este obligatoriu.

5. **-s numefişier** specifică fişierul în care se memorează rezultatul. Dacă argumentul opţiuni **-f** este un pattern, argumentul acestei opţiuni trebuie să fie de asemenea un pattern. Acest argument este obligatoriu.

6. **-a numefişier** specifică numele fişierului în care se va memora o listă de comenzi pentru `gnuplot` pentru ca să se poată vizualiza grafic (dacă acest lucru este posibil) rezultatele. Acest argument poate apare doar dacă argumentul opţiuni **-f** este un pattern. Dacă nu se specifică acest argument se consideră valoarea sa implicită `animation.gpl`.

7. **-r numefişier** specifică numele fişierului în care se memorează valorile medii pa'-tratice. Acest parametru poate fi folosit în condiţiile în care parametrul **-a** poate fi folosit şi doar dacă fişierul (sau fişierele) care conţine datele de intrare este de fapt fişier cu date de test (are pe ultima coloană valoarea corectă a ieşirii).

8. **-N** specifică faptul că inferenţa se face luînd în considerare primi n cei mai apropiaţi clusteri. Aceasta este varianta implicită.

9. **-n** specifică numărul de clusteri de luat în seamă dacă opţiunea **-N** este specificată. Dacă obţiunea nu este prezentă se consideră că parametrul are valoarea 3.

10. **-E** specifică faptul că la inferenţă se utilizează o funcţie gauss de determinare a ponderi fiecărei reguli. Dacă această opţiune este folosită opţiunile **-N** şi **-n** nu pot fi folosite şi viceversa.

11. **-e** specifică valoarea coeficientului care apare în funcţia gauss. Poate fi folosită doar dacă opţiunea **-E** este folosită.

7.2.3 Programul `mkfm`

Acest program generează o maşină quasiconstantă sau quasiliniară care poate fi folosită ca maşină iniţială pentru programul `clus`. Această maşină nu se obţine printr-un proces de învăţare cum s-ar putea crede ci încearcă să `ghicească` cu un efort computaţional minim una. Se pleacă de la presupunerea că alegerea iniţială a centrilor de greutate a clusterilor din r"andul datelor de antrenament este mai bună dec"at alegerea aleatoare a acestora.

Sursa programului se găseşte în fişierul `mkfm.cc`.

Opţiunile **-h, -g, -n, -S, -H** au aceiaşi semnificaţie ca în cazul programului `clus`. Opţiunea **-f numefişier** specifică fişierul în care maşina fuzzy creată va fi memorată.

7.2.4 Programul `mkdif`

Acest program este utilizat pentru a genera fişierul eroare definit ca diferenţa dintre datele de test şi aproximaţia acestora produsă cu o maşină care a învăţat un set de date de antrenament (posibil acelaşi set folosit ca date de test). Eroarea medie pătratică este de asemenea calculată, programul putînd fi folosit şi în acest scop.

Sursa programului se găseşte în fişierul `mkdif.cc`.

Programul are trei parametri de apel. Primul este fişierul cu date de test, al doilea este cel cu datele aproximate iar al treilea este numele fişierului în care se va memora diferenţa celor două.

7.2.5 Programul `gnuplot`

`gnuplot` este un utilitar disponibil pe mai toate platformele UNIX destinat reprezentărilor de funcţii. Nu intenţionăm o prezentare c"at de c"at completă a acestuia, aceasta putînd fi găsită în pagina de man şi în help-ul încorporat. Ce vom prezenta este modalitatea reprezentării funcţiilor cu care lucrează pachetul `clus`.

Se recomandă utilizarea acestui program din *XWindows*, reprezentarea fiind de o bună calitate.

Programul `infer` poate la cerere (vezi descrierea de mai sus) produce o secvenţă animată care are ca şi cadre stadii ale învăţării după fiecare epocă. Vizualizarea acestei secvenţe animate se face cu comanda:

```
$ gnuplot animation.gpl
```

unde `animation.gpl` este produs de `infer`.

Pentru reprezentarea fucţiilor individuale direct cu programul `gnuplot` se procedează în felul următor:

```
$ gnuplot
```

sistemul va răspunde cu:

```
        G N U P L O T
        Linux version 3.5
        patchlevel 3.50.1.17, 27 Aug 93
        last modified Fri Aug 27 05:21:33 GMT 1993

        Copyright(C) 1986 - 1993   Colin Kelley, Thomas Williams

        Send comments and requests for help to info-gnuplot@dartmouth.edu
        Send bugs, suggestions and mods to bug-gnuplot@dartmouth.edu

Terminal type set to 'x11'
gnuplot>
```

Dacă se doreşte reprezentarea funcţiilor de o singură variabilă la promptul de mai sus se răspunde cu:

```
gnuplot> plot "fig1.dta" with lines
```

unde `fig1.dta` este numele fişierului care conţine descrierea funcţiei de reprezentat. La comanda de mai sus sistemul răspunde deschizînd o fereastră *XWindows* în care se va desena funcţia şi de asemenea va apărea un nou prompt la care dacă se doreşte se poate introduce o linie asemănătoare cu ultima pentru reprezentarea altei funcţii.

C"and se doreşte părăsirea utilitarului `gnuplot` la prompt se răspunde cu:

```
gnuplot> quit
```

Dacă se doreşte reprezentarea funcţiilor de două variabile este necesară introducerea înainte de a reprezenta vreo funcţie a unei comenzi care indică faptul că reprezentările sunt parametrice:

```
gnuplot> set parametric
```

după care pot urma oric"ate comenzi de reprezentare de funcţii:

```
gnuplot> splot "fig2.dta" with lines
```

urmate în final de o comandă `quit`.

Reprezentările de funcţii cu o dimensiune şi cu două dimensiuni pot fi şi intercalate fără probleme.

7.3 Optimizări posibile ale implementării

Enumerăm mai jos un număr de posibile îmbunătăţiri ale implementării actuale, care odată utilizate ar putea mări substanţial viteza:

1. utilizarea unei aritmetice în virgulă fixă în locul celei în virgulă mobilă folosită în acest moment, ţinînd cont de faptul că o astfel de aritmetică ar fi suficientă problemei în cauză şi că pentru majoritatea procesoarelor folosite este considerabil mai rapidă

2. utilizarea unei alternative la metoda lui Jacobi de determinare a vectorilor şi valorilor proprii aşa cum să sugerat în secţiunea anterioară

3. construirea unui mecanism de transport al datelor mai eficient între obiecte, cel actual fiind destul de lent

4. rescrierea implementării în limbajul C

Scopul acestei lucrări fiind în primul r"and prezentarea unei idei şi nu neapărat implementare foarte eficientă a acesteia considerăm că o implementare foarte eficientă ar fi avut ca şi dezavantaj lipsa de claritate caracteristică optimizărilor.

Capitolul 8

Testare

În acest capitol vom face c"ateva experimente de învăţare utilizînd programul `clus`. Aceste experimente pot fi privite şi ca exemple de utilizare ale acestui program.

Pentru a putea urmări c"at de bine se face învăţarea vom utiliza doar funcţii reprezentabile grafic adică doar cu o intrare şi o ieşire sau cu două intrări şi o ieşire.

Vom utiliza 6 funcţii, trei dintre ele preluate din articolul [4] (8.1, 8.2 şi 8.3), pentru experimentele noastre. Vom putea face o comparaţie astfel cu rezultate obţinute de alţi cercetători. Formele analitice pentru cele 6 funcţii sunt:

1. funcţia F_1 reprezentată grafic în figura 8.1, considerată pe intervalul $[-3,3]\times[-3,3]$

$$F_1(x,y) = -8(x-y)e^{-x^2-y^2} \tag{8.1}$$

2. funcţia F_2 reprezentată grafic în figura 8.2, considerată pe intervalul $[-10,10]\times[-10,10]$

$$F_2(x,y) = 0.1(x^2+y^2)\frac{\sin x}{x} \tag{8.2}$$

3. funcţia F_3 reprezentată grafic în figura 8.3, considerată pe intervalul $[-3,3]\times[-3,3]$

$$F_3(x,y) = 3\sin^2 x \sin y \tag{8.3}$$

4. funcţia F_4 reprezentată grafic în figura 8.4, considerată pe intervalul $[0,1]\times[0,1]$

$$F_4(x,y) = 1+x+y+0.1(\zeta-0.5) \tag{8.4}$$

   unde ζ este o variabilă aleatoare cu valori în intervalul [0.1]. Această funcţie este un plan cu *zgomot* şi o vom folosi pentru a studia comportarea algoritmilor în prezenţa zgomotului.

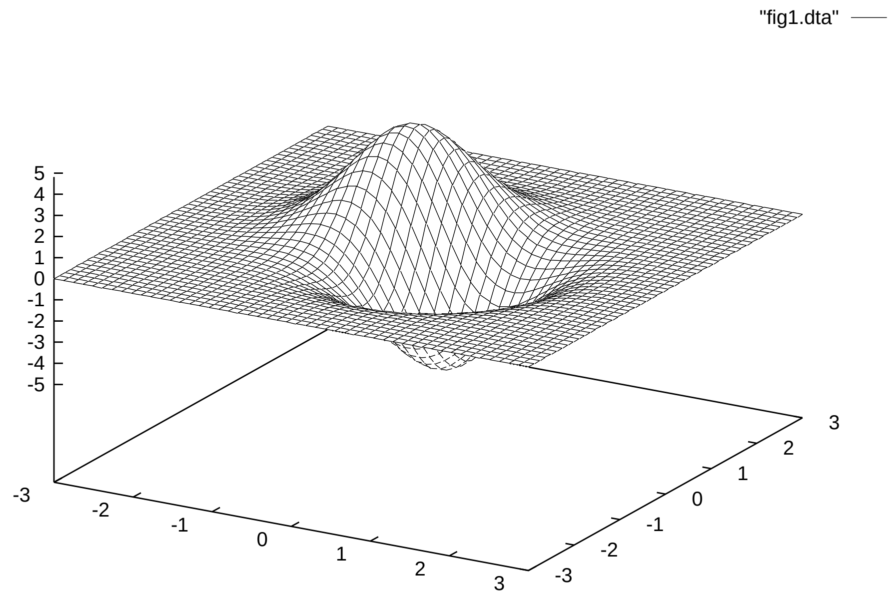


Figura 8.1: Reprezentarea grafică a funcţiei F_1

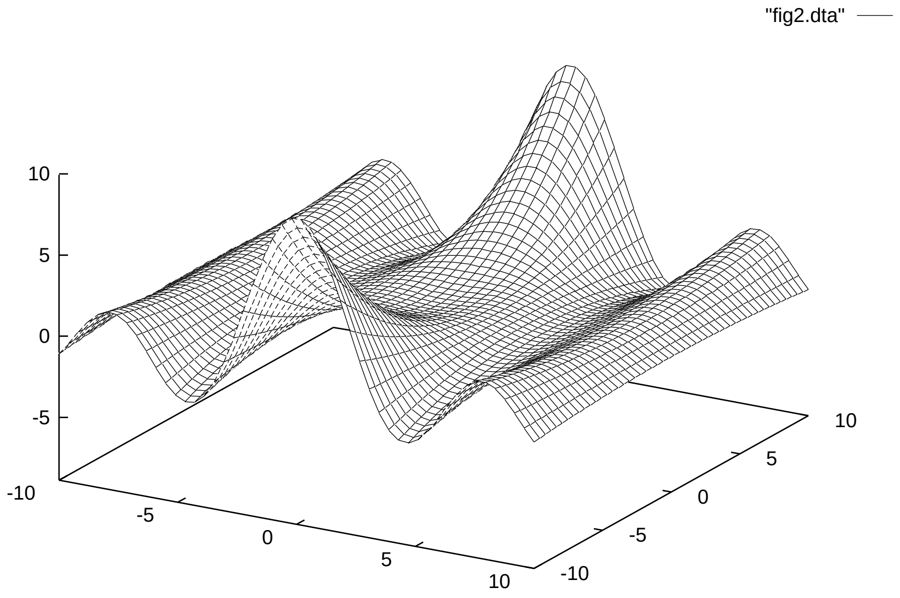


Figura 8.2: Reprezentarea grafică a funcţiei F_2

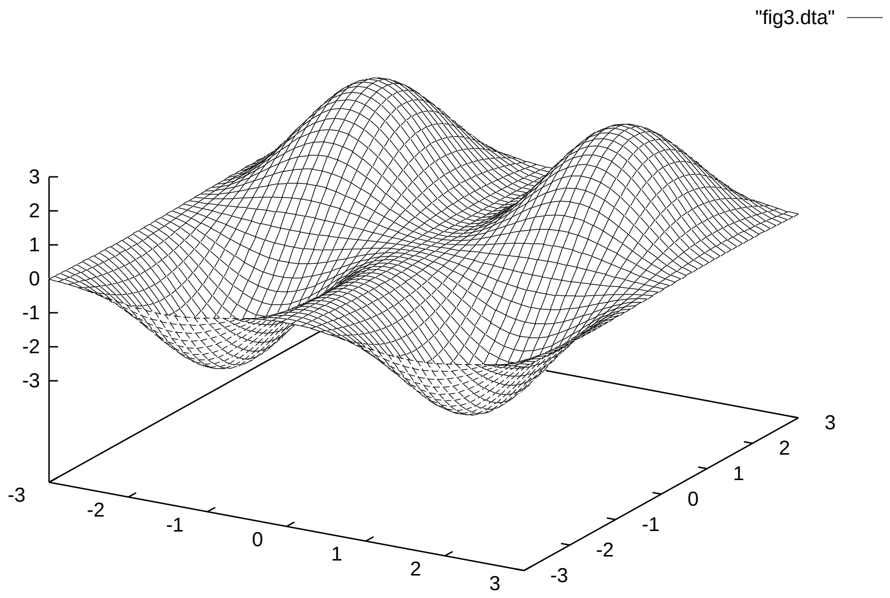


Figura 8.3: Reprezentarea grafică a funcţiei F_3

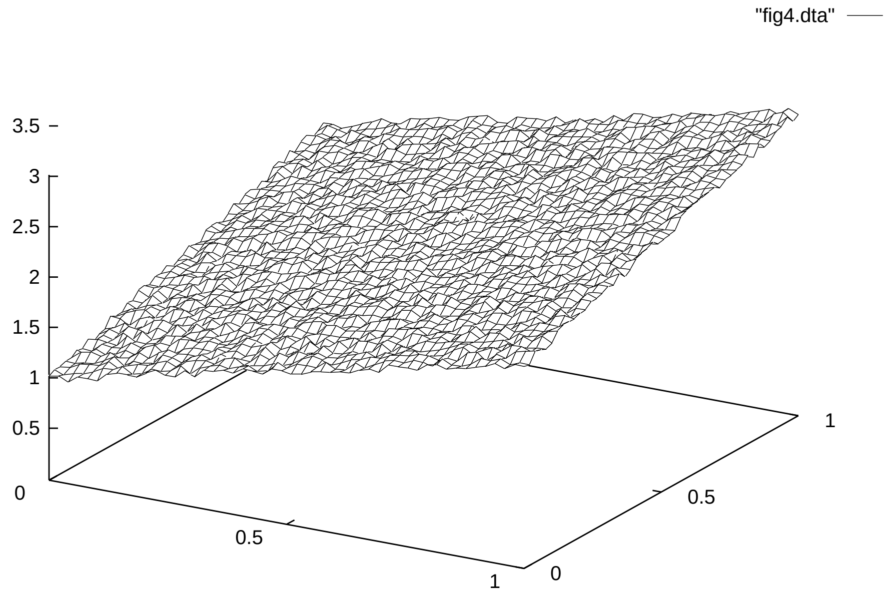


Figura 8.4: Reprezentarea grafică a funcţiei F_4

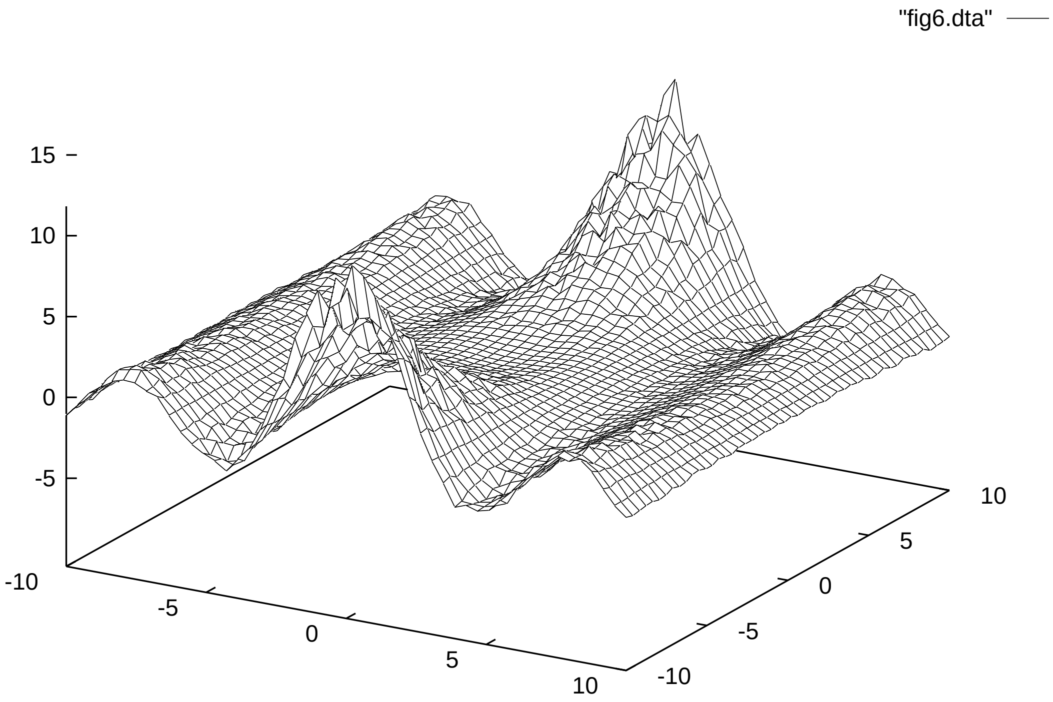


Figura 8.5: Reprezentarea grafică a funcţiei F_5

5. funcţia F_5 reprezentată grafic în figura 8.5, considerată pe intervalul $[-10, 10] \times [-10, 10]$
$$F_5(x, y) = 0.1(x^2 + y^2)\frac{\sin x}{x}(0.8 + 0.4\zeta) \tag{8.5}$$
unde ζ este o variabilă aleatoare cu valori în intervalul [0.1]. Funcţia este varianta cu *zgomot* a funcţiei 8.2 şi va fi de asemenea utilizată pentru a studia robusteţea algoritmilor.

În experimente se vor folosi valori ale funcţiilor $F_1, ..., F_5$ pe o grilă de 50×50 puncte.

Toate experimentele le vom efectua pe un calculator Pentium Intel 133 MHz care rulează un sistem de operare Linux 2.0.32. Toate reprezentările grafice sunt realizate cu programul `clus`.

În primele trei exemple vom reface experimentele din articolul [4] pentru a putea compara rezultatelor.

Experimentul 1 *Învăţăm funcţia F_1 folosind ca model o maşină quasiliniară cu 64 clusteri (aceiaşi arhitectură ca şi în articolul [4]). Vom genera maşinile după fiecare etapă pentru a putea trasa un grafic al evoluţiei erorii medii pătratice. Secvenţa de comenzi pentru a realiza acest experiment este:*

```
$ clus -g fig1.dta -p 1_64H_%d.fm -n 64 -H > J1_64H.dta
$ infer -g fig1.dta -f 1_64H_%d.fm -s rez1_64H_%d.dta -n 3 -N
  -r rmsn1_64H.hst
```

Reprezentăm acum grafic fişierele `rez1_64H_30.dta`*,* `J1_64H.dta` *şi* `rmsn1_64H.hst`*. Rezultatul se poate vedea în figurile 8.6, 8.7 a) şi b). Ultima valoare a erori medii pătratice este 0.0288 comparativ cu 0.0241 raportată în [4]. Trebuie să ţinem seama că eroarea medie pătratică nu este foarte potrivită pentru evaluarea acestui tip de învăţare. Timpul de rulare pentru cele 30 iteraţii ale procesului de învăţare a fost 94 secunde.*

Experimentul 2 *Facem un experiment asemănător cu cel precedent de data aceasta asupra funcţiei F_2. Comenzile folosite sunt:*

```
$ clus -g fig2.dta -p 2_64H_%d.fm -n 64 -H -i 50 > J2_64H.dta
$ infer -g fig2.dta -f 2_64H_%d.fm -s rez2_64H_%d.dta -n 3 -N
  -r rmsn2_64H.hst
```

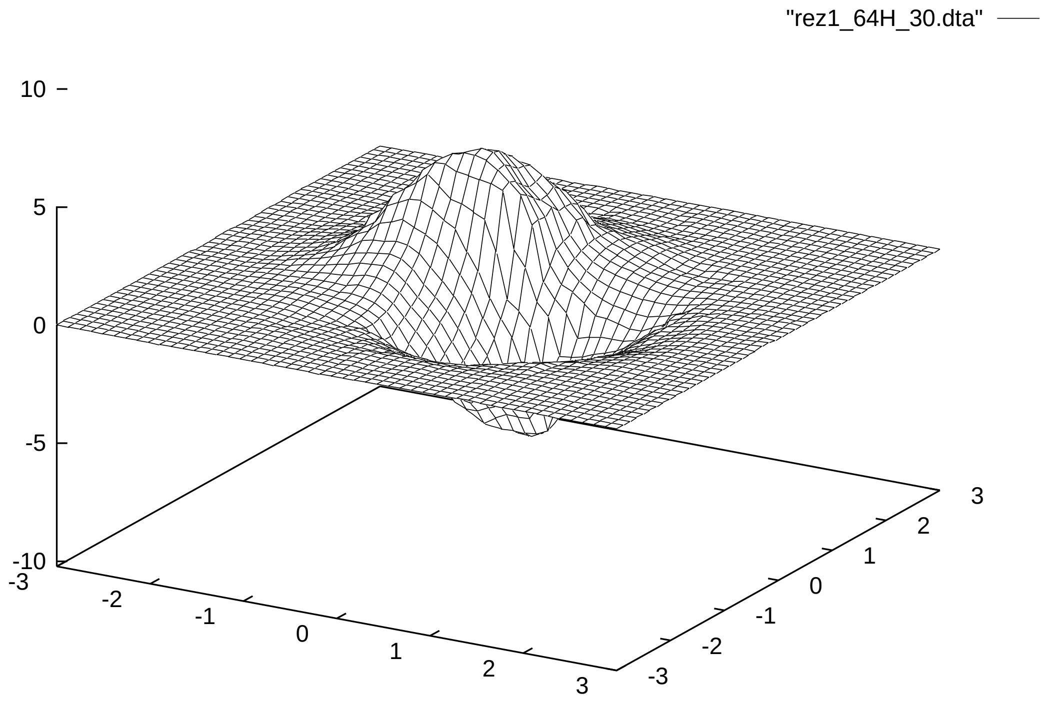
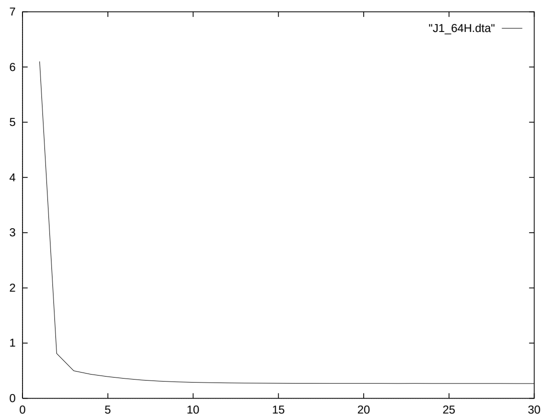
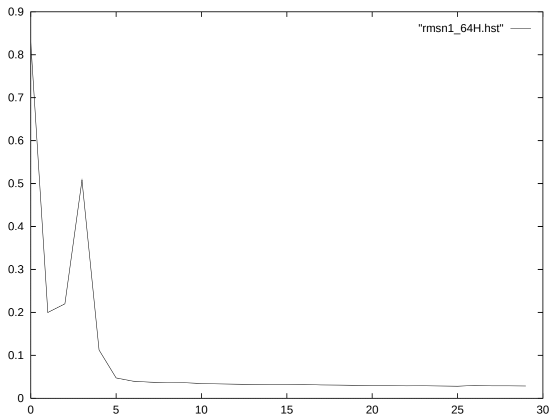


Figura 8.6: Aproximarea funcţiei F_1 folosind o maşină quasiliniară cu 64 clusteri (vezi experimentul 1)



a) Funcţia criteriu J



b) Eroarea medie paţratică

Figura 8.7: Evoluţia funcţiei criteriu şi a erorii medii pătratice în procesul de învăţare pt. F_1

Reprezentarea grafică a rezultatelor se găseşte în figurile 8.8, 8.9 a) şi b). Ultima valoare a erori medii pătratice este 0.0615 comparativ cu 0.0435 raportată în [4]. Timpul de rulare pentru cele 50 iteraţii ale procesului de învăţare a fost 153 secunde.

Experimentul 3 *Repetăm acum acelaşi tip de experiment pentru funcţia F_3. Lista de comenzi este:*

```
$ clus -g fig3.dta -p 3_64H_%d.fm -n 64 -H > J3_64H.dta
$ infer -g fig3.dta -f 3_64H_%d.fm -s rez3_64H_%d.dta -n 3 -N
  -r rmsn3_64H.hst
```

Reprezentarea grafică a rezultatelor se găseşte în figurile 8.10, 8.11 a) şi b). Ultima valoare a erori medii pătratice este 0.0479 comparativ cu 0.0105 raportată în [4]. Timpul de rulare pentru cele 30 iteraţii ale procesului de învăţare a fost 92 secunde.

După aceste trei experimente putem concluziona că deşi erorile medii pătratice sunt mai mari, numărul de epoci este considerabil mai mic (30 sau 50 în loc de 5000) pentru maşina quasiliniară comparativ cu datele publicate în articolul [4].

Continuarea învăţării după cele 30 epoci nu mai are sens deoarece nici valoarea funcţiei criteriu nici eroarea medie pătratică nu se mai modifică semnificativ, observîndu-se pentru aceasta chear o uşoară oscilaţie.

Vom repeta experimentul 3 pentru maşini quasiconstante. Alegem o maşină cu 110 clusteri pentru ca numărul de parametri să fie cam acelaşi cu al unei maşini quasiliniare cu 64 clusteri.

Experimentul 4 *Învăţăm funcţia F_3 utilizînd ca model o maşină quasiconstantă cu 110 clusteri. Ca şi în experimentul 1 generăm maşini după fiecare epocă. Comenzile necesare sunt:*

```
$ clus -g fig3.dta -p 3_110S_%d.fm -n 110 -S > J3_110S.dta
$ infer -g fig3.dta -f 3_110S_%d.fm -s rez3_110S_%d.dta -n 3 -N
  -r rmsn3_110S.hst
```

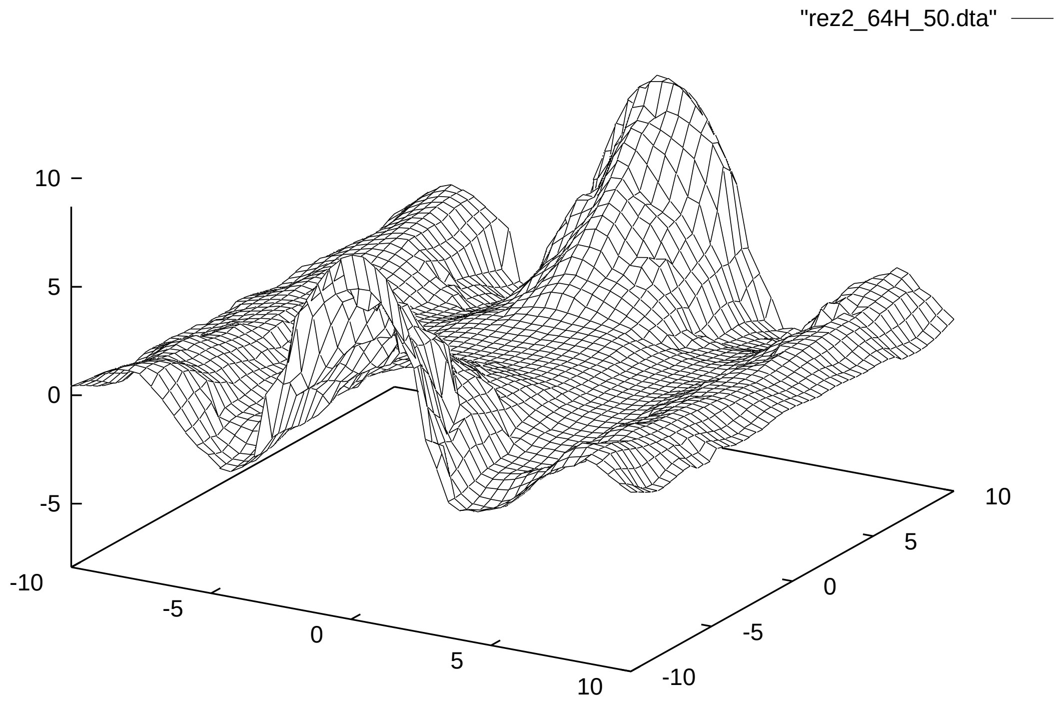
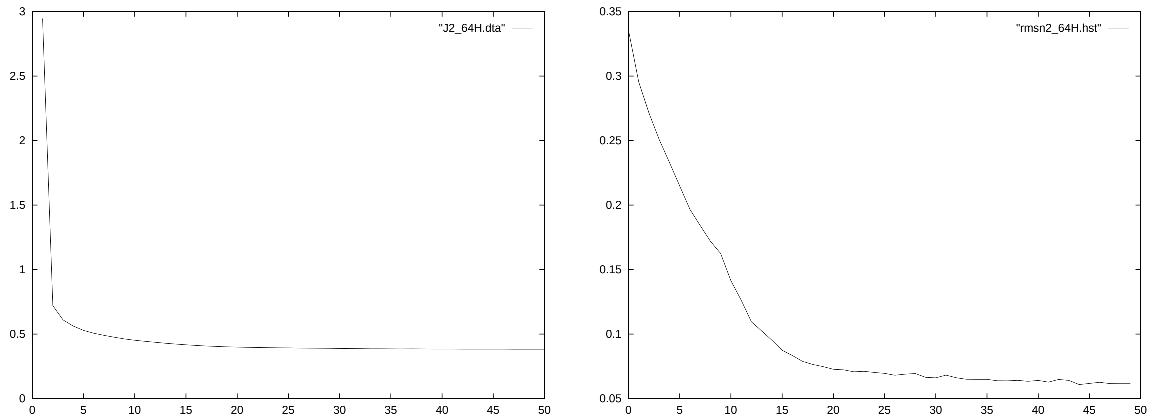


Figura 8.8: Aproximarea funcţiei F_2 folosind o maşină quasiliniară cu 64 clusteri (vezi experimentul 2)



a) Funcţia criteriu J b) Eroarea medie pătratică

Figura 8.9: Evoluţia funcţiei criteriu şi a erorii medii pătratice în procesul de învăţare pt. F_2

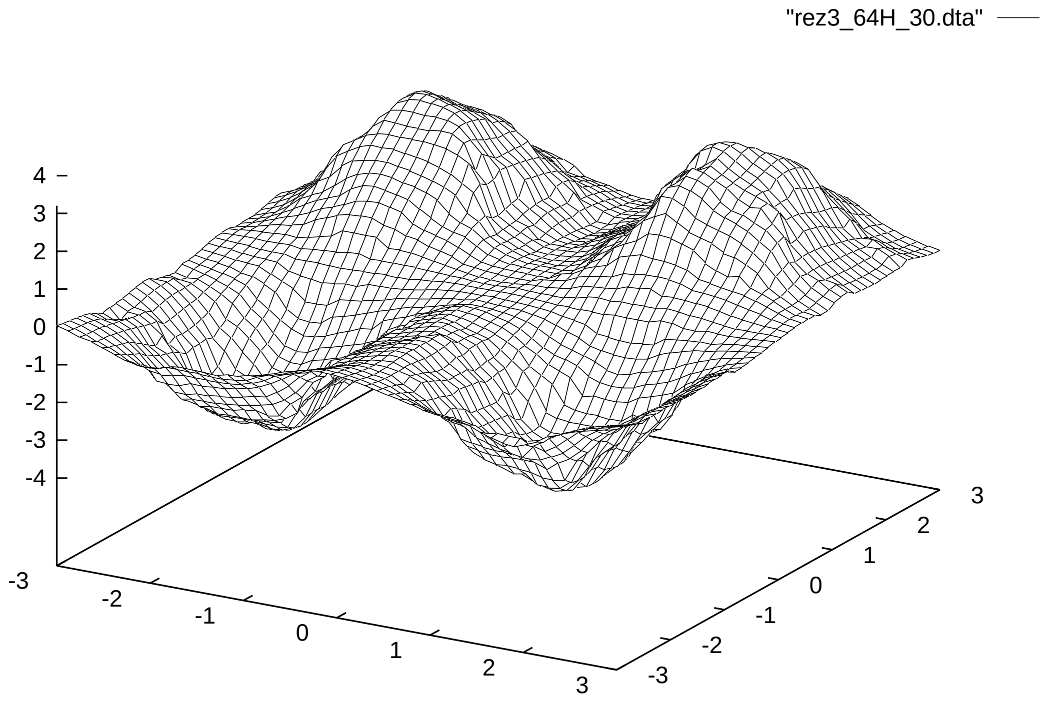
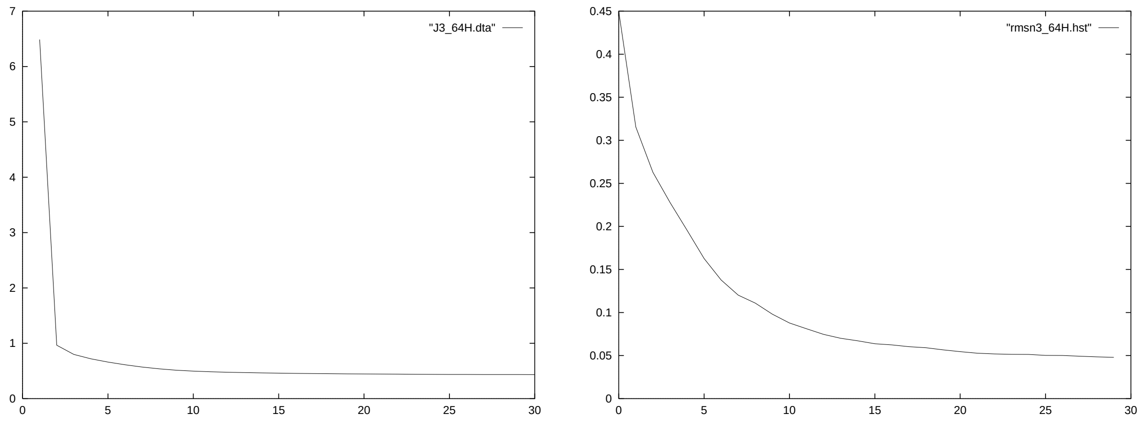


Figura 8.10: Aproximarea funcţiei F_3 folosind o maşină quasiliniară cu 64 clusteri (vezi experimentul 3)



a) Funcţia criteriu J b) Eroarea medie pătratică

Figura 8.11: Evoluţia funcţiei criteriu şi a erorii medii pătratice în procesul de învăţare pt. F_3

Reprezentarea grafică a rezultatelor se găseşte în figurile 8.12, 8.13 a) şi b). Ultima valoare a erori medii pătratice este 0.0709 comparativ cu 0.0479 pentru maşina quasiliniară. Timpul de rulare pentru cele 30 iteraţii ale procesului de învăţare a fost 44 secunde, jumătate din timpul necesar maşinii quasiliniare.

După cum se poate observa, deşi mult mai rapidă la acelaşi număr de parametrii, maşina quasiconstantă are rezultate mai slabe dec"at maşina quasiliniară. Rezultatele acestui experiment nu sunt singulare, toate testele efectuate de autor ducînd la aceleaşi concluzii.

Vom efectua acum două experimente care să ne ajute să tragem concluzii legate de comportarea în condiţii de zgomot a maşinii quasiliniare. Două funcţii afectate de zgomot aleator vor fi *învăţate.*

Experimentul 5 *Vom învăţa funcţia F_4 care este un plan cu zgomot folosind un model quasiliniar cu un singur cluster. Acest experiment ne va lămuri dacă maşina quasiliniară este capabilă să înveţe funcţii liniare. Secvenţa de comenzi necesară este:*

```
$ clus -g fig4.dta -p 4_1H_%d.fm -n 1 -H  > J4_1H.dta
$ infer -g fig4.dta -f 4_1H_%d.fm -s rez4_1H_%d.dta -n 3
  -N -r rmsn4_1H.hst
```

Reprezentarea grafică a rezultatelor se găseşte în figurile 8.14, 8.15 a) şi b). După cum se poate observa din graficul erorii medii pătratice, după doar o epocă maşina a şi învăţat c"at se poate de bine. Ecuaţia planului determinată este $y = 0.9987 + 0.9896x + 0.9929y$ foarte aproape de ecuaţia curată *a planului. Putem spune că maşinile quasiliniare se descurcă foarte bine cu funcţiile liniare chear dacă acestea sunt afectate de zgomot.*

Experimentul 6 *Acest experiment probează puterea de generalizare a maşini quasiliniare. Vom învăţa funcţia F_5 care este versiunea cu zgomot a funcţiei F_2 folosind ca mode o maşină quasiliniară cu 64 clusteri. 'In loc să calculăm eroarea medie pătratică faţă de funcţia F_5 o vom calcula faţă de F_2. În comenzi aceasta se transcrie:*

```
$ clus -g fig6.dta -p 6_64H_%d.fm -n 64 -H -i 50 > J6_64H.dta
$ infer -g fig2.dta -f 6_64H_%d.fm -s rez6_64H_%d.dta -n 3 -N
  -r rmsn6_64H.hst
```

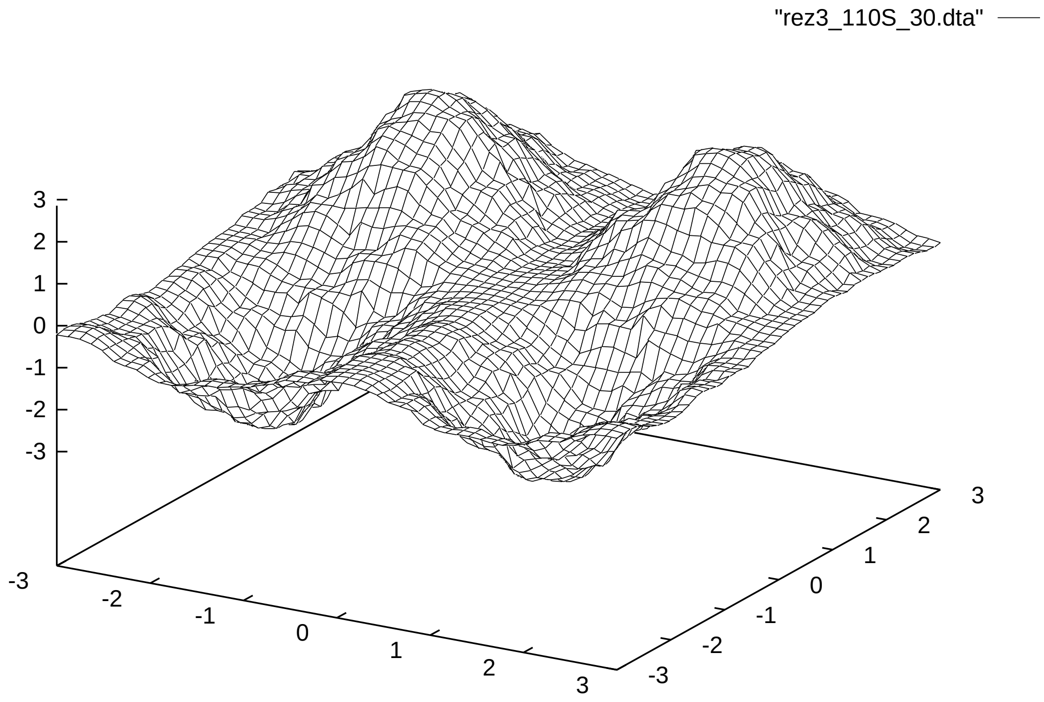
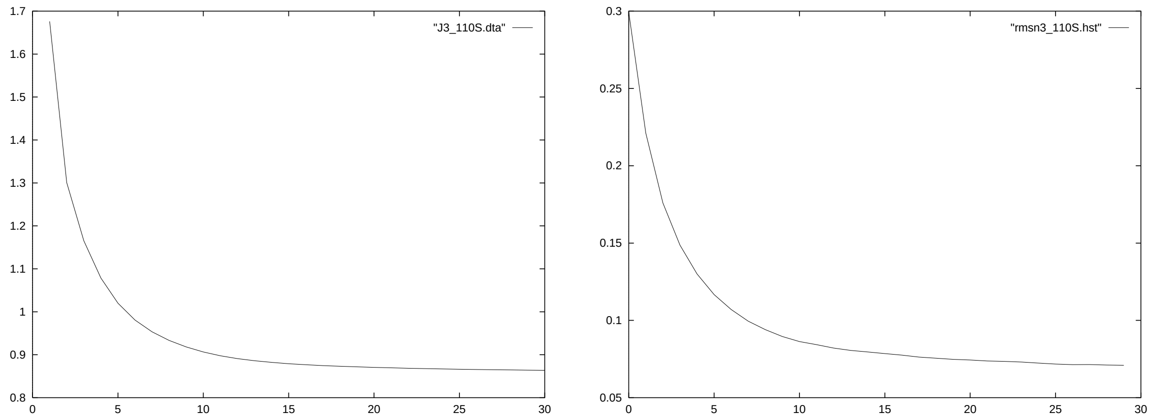


Figura 8.12: Aproximarea funcţiei F_3 folosind o maşină quasiconstantă cu 110 clusteri (vezi experimentul 4)



a) Funcţia criteriu J

b) Eroarea medie pătratică

Figura 8.13: Evoluţia funcţiei criteriu şi a erorii medii pătratice în procesul de învăţare pt. F_3

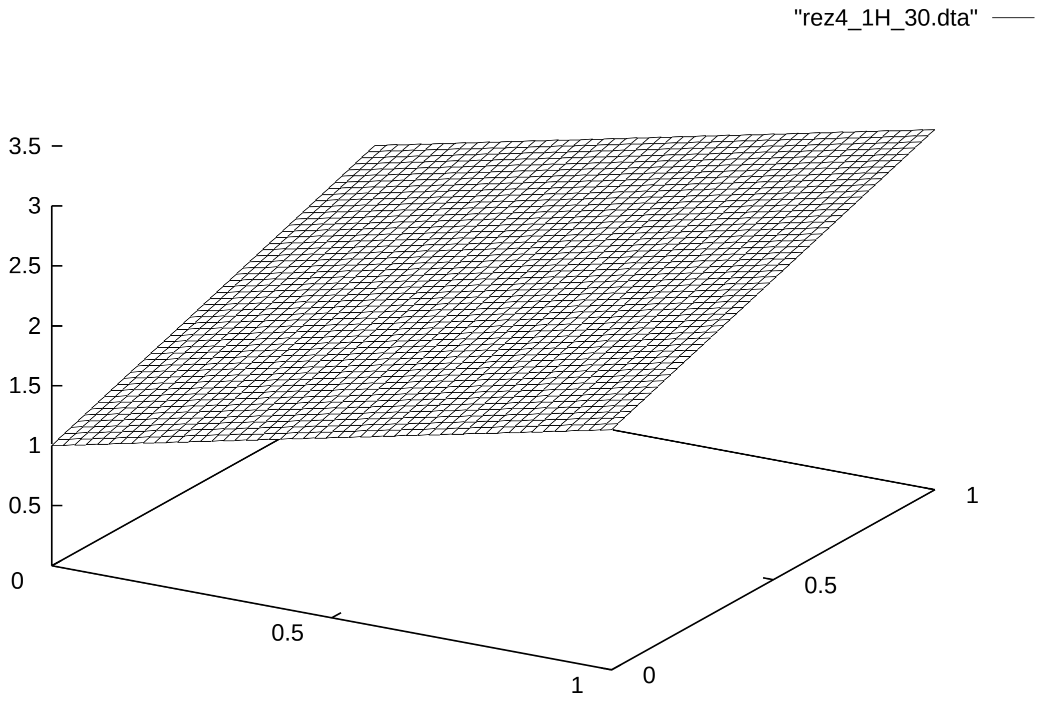
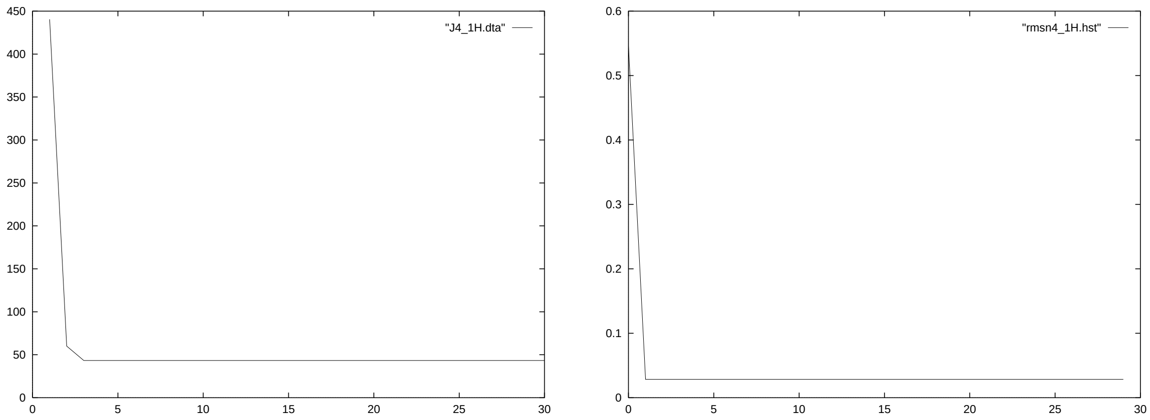


Figura 8.14: Aproximarea funcţiei F_4 folosind o maşină quasiliniară cu un cluster (vezi experimentul 5)



a) Funcţia criteriu J b) Eroarea medie pătratică

Figura 8.15: Evoluţia funcţiei criteriu şi a erorii medii pătratice în procesul de învăţare pt. F_3

Reprezentarea grafică a rezultatelor se găseşte în figurile 8.16, 8.17 a) şi b). Ultima valoare a erori medii pătratice este 0.0615 comparativ cu 0.0435 c"at a fost obţinut pentru experimentul 2. Timpul de rulare pentru cele 50 iteraţii ale procesului de învăţare a fost 153 secunde.

Din experimentul de mai sus putem concluziona că algoritmul se comportă satisfăcător de bine în prezenţa zgomotului, neexistînd diferenţe mari între rezultatele obţinute în acesta vizavi de cele obţinute în experimentul 2.

În continuare vom încerca să utilizăm metoda sugerată în capitolul 6 de construire a unei maşini iniţiale pentru procesul de învăţare luînd ca centri ai clusterilor puncte din datele de intrare-ieşire. Vom efectua două experimente, unul cu maşini quasiconstante şi altul cu maşini quasiliniare.

Experimentul 7 *Vom relua învăţarea funcţiei F_3, utilizînd de data aceasta iniţializarea de care vorbeam, folosind o maşină quasiconstantă tot cu 110 clusteri. Ne interesează comparativ cu experimentul 4 cum evoluează procesul de învăţare. Comenzile folosite sunt:*

```
$ mkfm -g fig3.dta -f i3_110S.fm -n 110 -S
$ clus -g fig3.dta  -f i3_110S.fm -p 3_110Si_%d.fm  > J3_110Si.dta
$ infer -g fig3.dta -f 3_110Si_%d.fm -s rez3_110Si_%d.dta -n 3 -N
  -r rmsn3_110Si.hst
```

Reprezentarea grafică a rezultatelor se găseşte în figurile 8.18, 8.19 a) şi b). Ultima valoare a erori medii pătratice este 0.0688 comparativ cu 0.0709 pentru maşina quasiconstantă obţinută în condiţiile experimentului 4. La reprezentarea erorilor am pus pe acelaşi grafic şi valorile obţinute în experimentul 4 pentru a putea face o comparaţie. După cum se poate observa se economisesc cam 5 epoci de antrenament folosind această metodă de iniţializare.

Experimentul 8 *Refacem experimentul 3 folosind iniţializarea maşinii fuzzy. Utilizăm o maşină quasiliniară cu 64 clusteri. Secvenţa de comenzi este:*

```
$ mkfm -g fig3.dta -f i3_64H.fm -n 64 -H
$ clus -g fig3.dta  -f i3_64H.fm -p 3_64Hi_%d.fm  > J3_64Hi.dta
```

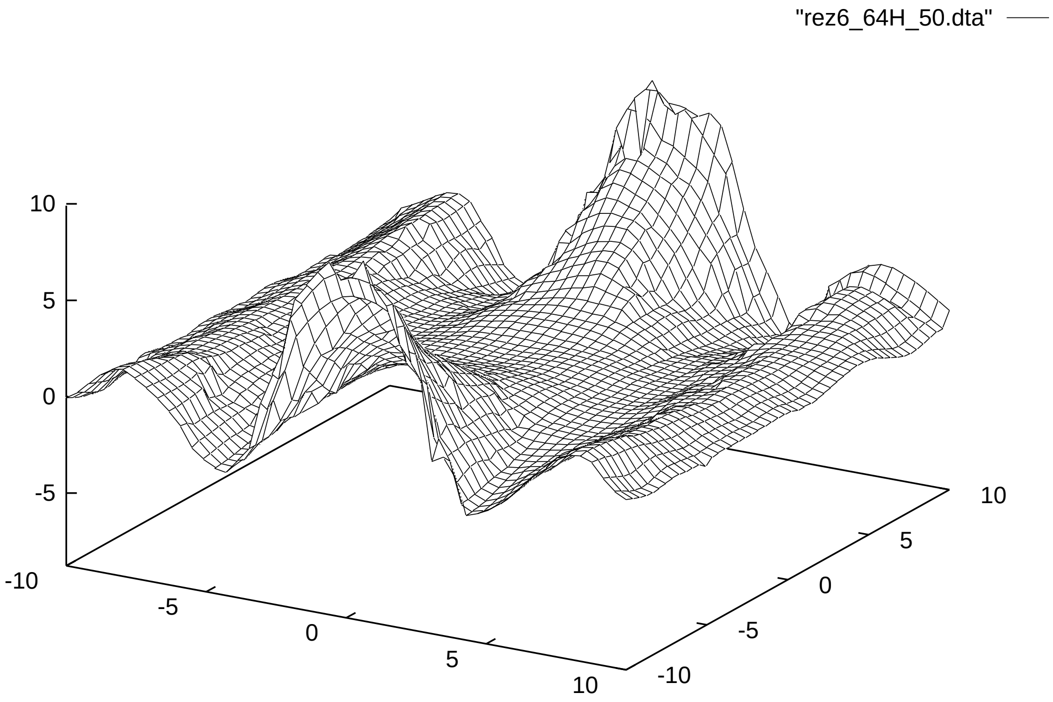
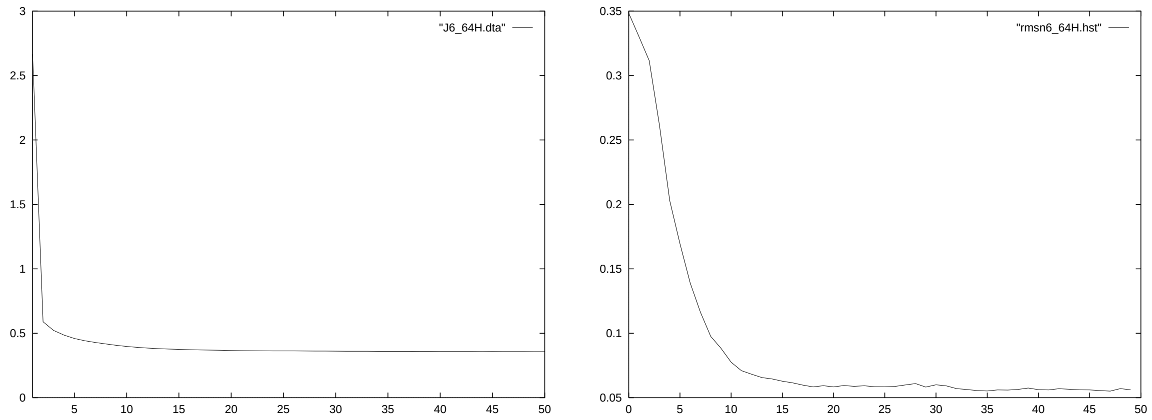


Figura 8.16: Aproximarea funcţiei F_5 folosind o maşină quasiliniară cu 64 clusteri (vezi experimentul 6)



a) Funcţia criteriu J b) Eroarea medie pătratică

Figura 8.17: Evoluţia funcţiei criteriu şi a erorii medii pătratice în procesul de învăţare pt. F_5

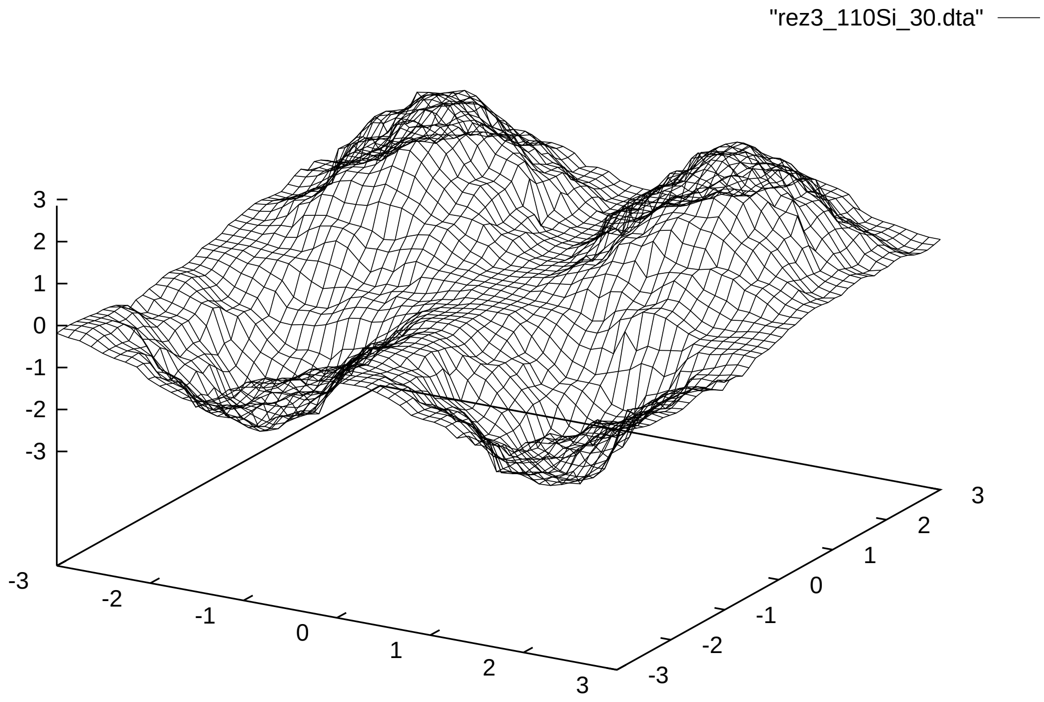
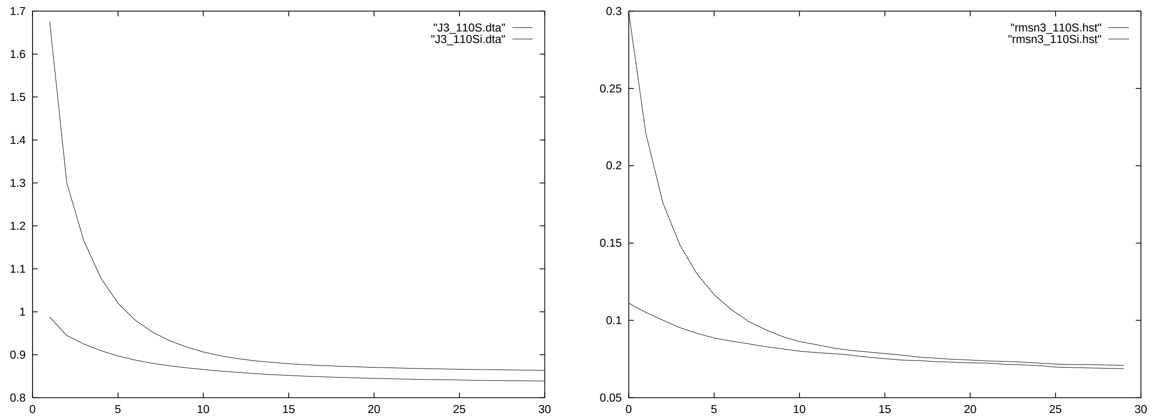


Figura 8.18: Aproximarea funcţiei F_3 folosind o maşină quasiconstantă cu 110 clusteri (vezi experimentul 7)



a) Funcţia criteriu J

b) Eroarea medie pătratică

Figura 8.19: Evoluţia funcţiei criteriu şi a erorii medii pătratice în procesul de învăţare pt. F_3

```
$ infer -g fig3.dta -f 3_64Hi_%d.fm -s rez3_64Hi_%d.dta -n 3 -N
  -r rmsn3_64Hi.hst
```

Reprezentarea grafică a rezultatelor se găseşte în figurile 8.18, 8.19 a) şi b). La reprezentarea erorilor am pus pe acelaşi grafic şi valorile obţinute în experimentul 4 pentru a putea face o comparaţie. După cum se poate observa se economisesc cam 6 epoci de antrenament folosind această metodă de iniţializare.

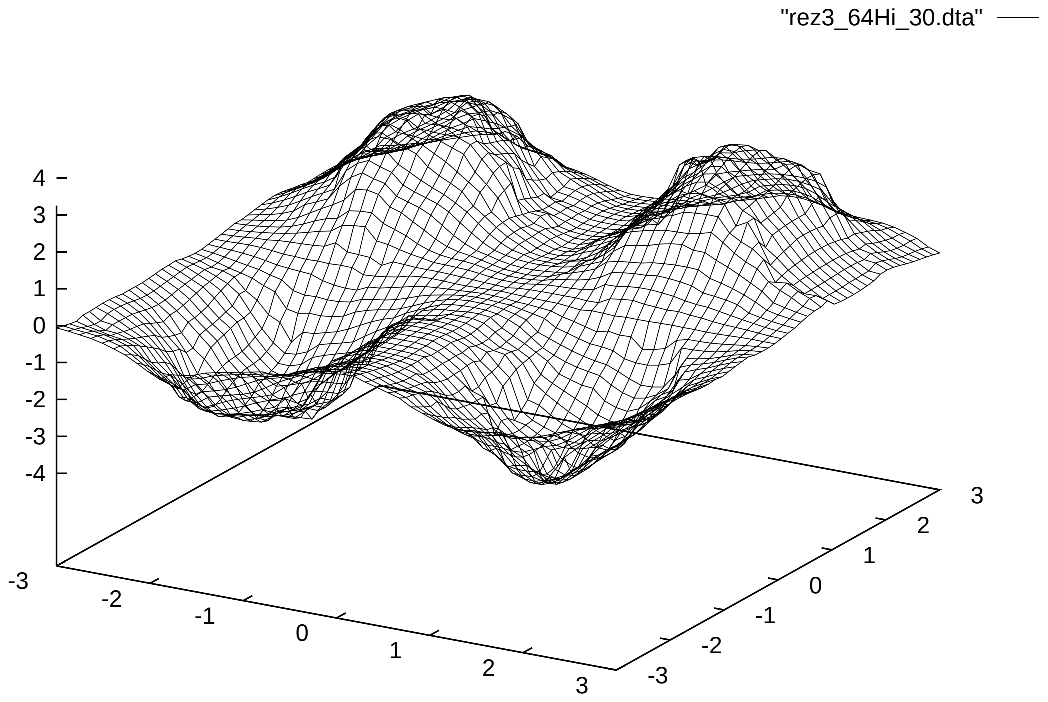
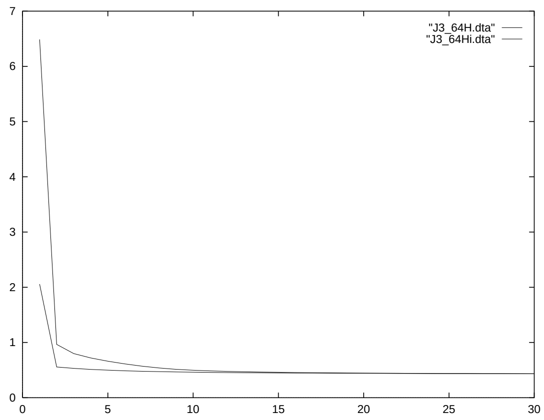
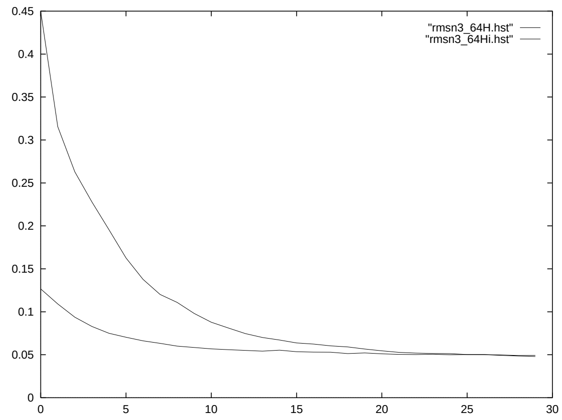


Figura 8.20: Aproximarea funcţiei F_3 folosind o maşină quasiliniară cu 64 clusteri (vezi experimentul 3)



a) Funcţia criteriu J



b) Eroarea medie pătratică

Figura 8.21: Evoluţia funcţiei criteriu şi a erorii medii pătratice în procesul de învăţare pt. F_3

Bibliografie

[1] M. Balazinski, E. Czogala, and T. Sadowski. Modelling of neural.

[2] A. Blanco and M. Delgado. A direct fuzzy inference procedure by neural networks.

[3] J. Buckley. Sugeno type controllers are universal controllers. *Fuzzy Sets and Systems*, (53):299–303, 1993.

[4] M. Chelaru and C. Dumitrascu. Sisteme fuzzy sugeno diferenţiale.

[5] C.-L. Chen, P.-C. Chen, and C.-K. Chen. Analysis and design of fuzzy control system. *Fuzzy Sets and Systems*, (57):125–140, 1993.

[6] I. Corovei. Algebră. Notiţe de curs.

[7] A. Dobra. Identification of fuzzy models using clustering techniques. In *Automatic control and testing conference*, pages 285–290, 1996.

[8] A. Dumitraş. *Proiectare reţelelor neuronale artificiale*. ODEON, 1997.

[9] D. Dumitrescu. *Teoria clusterizării*. Universitatea Babes-Bolyai, 1991.

[10] D. Dumitrescu. *Inteligenţă artificială*. Universitatea Babes-Bolyai, 1995.

[11] D. Dumitrescu. *Modele conexioniste în inteligenţa artificială*. Universitatea Babes-Bolyai, 1995.

[12] D. Dumitrescu and C. Hariton. *Reţele neuronale*. TEORA, 1996.

[13] D. Filev. Fuzzy modeling of complex systems. *International Jurnal of Approzimate Reasoning*, (5):281–290, 1991.

[14] V. Iorga and B. Jora. *Progamare numerică*. Teora, 1996.

[15] D. Isoc. *Contribuţii privind studiul şi implementarea sistemelor de control inteligent cu logică fuzzy în procese ale industriei textile*. PhD thesis, Universitatea Tehnică Cluj-Napoca, 1993.

[16] E. Mamdani. Applications of fuzzy algorithms for controll of simple dynamic plant. *Proc. IEE*, (121):1585–1588, 1974.

[17] M. Postolache. *Metode numerice*. Editura SIRIUS, 1993.

[18] M. Sugeno and G. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, (28):15–33, 1988.

[19] M. Sugeno and K. Tanaka. Successive identification of a fuzzy model and its applications to prediction of a complex system. *Fuzzy Sets and Systems*, (42):315–334, 1991.

[20] T. Takagi and M. Sugeno. Fuzzy identification of systems ans its applications to modeling and control. *IEEE Trans. Systems Man Cybernet.*

[21] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and controll. *IEEE Trans. Systems Man Cybernet*, (15):116–132, 1985.

[22] G. Toderean, M. Coşteiu, and M. Giurgiu. *Reţele neuronale*. Microinformatica, 1994.

Listingul programului

În continuare prezentăm listingul programului `clus`. Deşi a fost integral implementat pe o platformă *Linux* nu conţine secvenţe de cod dependente de maşină. Acest lucru face ca acest program să poată fi compilat pe orice platformă pe care există un compilator standard de C++ fiind eventual necesare modificări ale fişierului `Makefile`.