UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU
FACULTATEA DE INGINERIE
DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

PROIECT DE DIPLOMĂ

Îndrumător științific : Conf. dr. ing. Morariu Daniel

Absolvent: Badiu Bogdan-Vasile
Specializarea: Ingineria Sistemelor Multimedia

-Sibiu, 2021-

UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU
FACULTATEA DE INGINERIE
DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

Studiu comparativ între diferiți algoritmi de clustering

Îndrumător științific : Conf. dr. ing. Morariu Daniel

Absolvent: Badiu Bogdan-Vasile
Specializarea: Ingineria Sistemelor Multimedia

Cuprins

1 Prezentarea Temei 3

2 Considerații teoretice 5

2.1 Algoritmi de învățare 5

2.1.1 Categorii de algoritmi de învățare 6

2.1.1.1 Algoritmi de învățare supervizată 6

2.1.1.2 Algoritmi de învățare nesupervizată 6

2.1.1.3 Algoritmi de învățare semi-supervizată 6

2.1.2 Modul de învățare 7

2.1.2.1 Modul de învățare Off-line 7

2.1.2.2 Modul de învățare On-line 7

2.1.3 Etapele procesului de învățare 7

2.1.3.1 Etapa de antrenare 7

2.1.3.2 Etapa de testare 7

2.2 Metode de normalizarea a datelor 8

2.2.1 Normalizarea binară 8

2.2.2 Normalizarea nominală 8

2.2.3 Normalizarea Suma1 9

2.2.4 Normalizarea min-max 9

2.2.5 Normalizarea Cornell-Smart 9

2.3 Metode de calcul a similarității 10

2.3.1 Tipuri de distanțe 11

2.3.1.1 Distanța Manhattan 11

2.3.1.2 Distanța Euclidiană 11

2.3.1.3 Distanța Minkowski 11

2.3.1.4 Distanța Cosinus 11

2.3.2 Matricea de similaritate 12

2.4 Algoritmi de clustering 12
2.4.1 Generalități 13
2.4.1.1 Cerințele algoritmilor de clustering 13
2.4.1.2 Clasificarea metodelor de clustering 14
2.4.2 Algoritmul DBSCAN 16
2.4.3 Algoritmul HAC (Hierarchical agglomerative clustering) 19
2.4.3.1 Single link 21
2.4.3.2 Complete link 21
2.4.3.3 Average link 22
2.4.3.4 Centroid link 22
2.5 Metode de evaluare 23
2.5.1 Măsuri externe de evaluare 24
2.5.2 Măsuri interne de evaluare 25
3 Considerații practice 27
3.1 Structura aplicației realizate 27
3.2 Etapa de citire a datelor 30
3.3 Etapa de normalizare a datelor 37
3.4 Etapa de calcul a matricei de similaritate 42
3.5 Etapa de clustering 46
3.5.1 DBSCAN 46
3.5.2 HAC (Hierarchical agglomerative clustering) 49
3.6 Etapa de atribuire a unei clase pentru un cluster 53
3.7 Etapa de evaluare 58
3.8 Rezultate experimentale 63
3.9 Interfața aplicației 74
4 Concluzii 78
5 Bibliografie 79

1 Prezentarea Temei

Algoritmii diferiți de clustering grupează diferit același set de date. Aceștia sunt influențați de metoda pe care se bazează algoritmul și de forma datelor de intrare. Același algoritm de clustering poate grupa diferit datele, deoarece acesta este influențat de metrica de similaritate pe care o folosește și de parametri de intarea pe care trebuie să îi primească algoritmul.

Scopul acestei lucrări este acela de a face o analiză a algoritmilor de clustering, pe baza metricilor externe, prin care să se determine care algoritm este mai eficient și în ce context. Este important de speificat că algoritmi de clustering au avantaje și dezavantaje, fiecare algoritm fiind specializat în rezolvarea anumitor probleme. Cu ajutorul acestei teme se încearcă diferențierea algoritmilor de clustering și cum trebuie aceștia aleși în funcție de necesitate.

Analiza algoritmilor se face din mai multe perspective. Se analizează cum sunt clusterizate datele după aplicarea algoritmului de clustering și se compară rezultatele, astfel încât să se determine care algoritm este mai eficient. Se face o analiză și a performanței algoritmilor și în funcție de datele de la intrare și de parametrii pe care algoritmul îl primește. Datele de intrare sunt normalizate cu mai multe metode, astfel încât să se determine ce metodă de normalizare este mai bună pentru fiecare algoritm. Parametrii de intrare pot influența algoritmii de clustering, de aceea se face o analiză a cum sunt influențați algoritmii în funcție de acești parametri. Algoritmii de clustering sunt influențați și de modul în care este calculată similaritatea dintre date. Similaritate dintre date este calculată cu ajutorul distanțelor, de aceea se folosesc mai multe metode de calcul al distanțelor pentru a analiza ce metodă de calcul este mai bună pentru fiecare algoritm.

Această temă propune realizarea unui proces prin care să se aplice diferiți algoritmi de clustering asupra datelor. Procesul trebuie să conțină mai multe etape cu ajutorul cărora se poate realiza o analiză pe baza unor metrici și pe baza acestor analize să se compare algoritmii de clustering. Acest proces face comparația între doi algoritmi diferiți de clustering, algoritmul DBSCAN și algoritmul HAC (Hierarchical agglomerative clustering).

Pentru a realiza acest proces am implementat mai multe metode de calcul a distanțelor, mai multe metode de normalizare a datelor și cei doi algoritmi care o să fie comparați la finalul procesului. La finalul procesului am realizat o evaluare cu ajutorul metricilor, iar cu ajutorul acestor evaluări se compară cei doi algoritmi.

Lucrarea este împărțită în două capitole importante cu ajutorul cărora se încearcă descrierea și înțelegerea fiecărei etapă pe care procesul de clustering o implementează.

Primul capitol important este capitolul „Considerații teoretice” care ajută la o înțelegere mai bună a fiecărei etape în parte. Acest capitol este împărțit în mai multe subcapitole care descriu în

detaliu metodele folosit în fiecare etapă. Primul subcapitol descrie algoritmii de învățare automată, unde sunt aceștia folosiți și care sunt categoriile din care fac parte aceștia. Al doilea subcapitol descrie normalizare, de ce este aceasta folosită și câteva metode folosite. Următorul subcapitol descrie similaritatea, cum este aceasta calculată și câteva metode cu ajutorul cărora se calculează. Subcapitolul patru descrie algoritmii de clustering, cum sunt aceștia împărțiți și oferă face o descriere detaliată a celor doi algoritmi folosiți. Ultimul subcapitol din acest capitol prezintă și descrie categoriile de evaluare a algoritmilor de învățare.

Al doilea capitol important este capitolul de „Considerații practice" prin care se arată importanța fiecărei etape a procesului și cum a fost implementată aceasta. Primul subcapitol prezintă etapele procesului și structura aplicației care realizează procesul. Al doilea subcapitol prezintă etapa de citire a datelor, importanța acestei etape și cum a fost aceasta implementată. Al treilea subcapitol arată cum a fost implementată etapa de normalizare a datelor și scopul acesteia. Al patrulea subcapitol prezintă cum a fost calculată similaritatea între date și cum a fost implementată fiecare metodă. Următoarele două subcapitole prezint implementările folosite pentru a face evaluarea datelor. În subcapitolul șase sunt prezentate câteva rezultate obținute după aplicarea algoritmilor, iar în ultimul subcapitol se face o scurtă descriere a interfeței aplicației care realizează acest proces.

La finalul lucrării sunt prezentate câteva concluzii la care s-a ajuns după aplicarea procesului de clustering.

Am ales această temă, deoarece am dorit să înțeleg cât mai multe despre algoritmii de învățare automată, algoritmii de clustering și cum se determină care algoritm este mai bun. Deși la început am descoperit că nu este o temă ușoară am privit-o ca pe o provocare. Cu ajutorul acestei teme am aflat că aplicarea algoritmilor de învățare automată trebuie să parcurgă mai multe etape și că este un proces mult mai complex decât credeam. Am descoperit cât de importantă este analiza datelor în multe domenii. Această temă mi-a arătat cât de importante sunt toate etapele prin care trebuie să treacă datele și cum acestea pot influența tot procesul.

2 Considerații teoretice

2.1 Algoritmi de învățare

Învățarea automată este un subdomeniu al Inteligenței Artificiale care se referă la capacitatea unei mașini de calcul de a imita inteligența umană. Aceasta se referă la capacitatea de a lua o decizie pe cont propriu pe baza unor experiențe anterioare.

Tehnicile de învățare automată sunt tehnici care combină metode fundamentale din știința calculatoarelor cu idei din probabilități și statistică. Aceste tehnici sunt utilizate pentru a se rezolva diferite tipuri de probleme, în ceea ce urmează se vor prezenta câteva tipuri de probleme:

- **Clasificarea documentelor sau textului:** acestea determină automat topicul din care face parte un text sau un document.
- **Data Mining**: acestea se referă la extragerea de cunoștințe din datele stocate în bazele de date.
- **Procesare de imagini:** acestea includ recunoașterea sau identificarea obiectelor din imagini, dar și recunoașterea sau detectarea de fețe sau emoții din imagini.
- **Aplicații de procesare de discursuri:** acestea încearcă să realizeze scrierea discursurilor, cuvintelor pe baza unei limbi sau a unei ceea ce se aude. Aici sunt incluse aplicațiile care recunosc discursul, interlocutorul și verifică discursul.
- **Aplicații medicale:** pe baza acestora se pot detecta multe tipuri de probleme de sănătate sau se pot face analize genetice.
- **Jocuri:** acestea pot juca singure jocuri pregum șah, table, GO, etc. [6]
- **Pentru controlul automat al mașinilor sau roboților:** acestea se referă la aplicațiile din automotive sau la cele din anumite zone de producție, de exemplu recunoașterea automată a produselor neconforme.

După cum este menționat și în [6] , învățarea automată poate fi definită ca aplicarea anumitor algoritmi care, pe baza experienței, îmbunătățesc performanța sau realizează predicții cât mai exacte. Acești algoritmi ajută mașina de calcul să ia o decizie pe cont propriu. În acest context, experiența se referă la informațiile învățate anterior de către algoritmi.

Informațiile din care se învață sunt extrase din date colectate și analizate. Datele pot fi sub forma de seturi de date etichetate sau pot fi obținute din interacțiunea cu mediul.

Algoritmii de învățare automată sunt algoritmi care transformă datele sau informațiile în modele pe care le poate învăța. Performanța acestora depinde de datele pe care le utilizează, astfel

se poate observa că învățarea automată depinde într-o mare măsură de anumite metode prin care se fac analize pe date și de tehnici statistice. [6]

Algoritmii de învățare automată sunt împărțiți în două categorii principale: algoritmi de învățare supervizată și algoritmi de învățare nesupervizată.

2.1.1 Categorii de algoritmi de învățare

Există mulți algoritmi de învățare, aceștia având la rândul lor mai multe tipuri de implementare, dar se poate face o clasificare a acestor algoritmi pe baza metodei de învățare pe care aceștia o folosesc. [4]

2.1.1.1 Algoritmi de învățare supervizată

Algoritmii de învățare supervizată se mai numesc și algoritmi de clasificare sau de predicție. Acest tip de algoritmi primește la intrare atât setul de date, cât și etichetele din care fac parte exemplele din acesta, încercând să învețe de ce anumite exemple din setul de date fac parte din anumite categorii sau de ce au acele etichete. [6]

Sarcina acestui tip de învățare este de a pune fiecare exemplu într-o anumită categorie pe care acesta o cunoaște de la intrare, adică în momentul în care apare un exemplu nou, algoritmul trebuie să specifice din ce categorie face parte exemplul sau ce etichetă are acesta. Această sarcină se numește clasificare.

2.1.1.2 Algoritmi de învățare nesupervizată

Algoritmii de învățare nesupervizată se mai numesc și algoritmi de clustering. Aceștia primesc la intrare doar setul de date, fără să se mai specifice etichetele din care fac parte exemplele din setul de date. Aceștia trebuie să grupeze exemplele în clusteri cât mai relevanți [4]. Se numește învățare nesupervizată, deoarece algoritmul nu primește ieșirea la care acesta trebuie să ajungă, el trebuie să grupeze exemplele în clusteri pe baza atributelor fiecărui exemplu.

2.1.1.3 Algoritmi de învățare semi-supervizată

În unele cazuri poate să apară și conceptul de învățare semi-supervizată. Acesta a rezultat din combinarea celor două moduri de învățare prezentate anterior: învățarea supervizată și învățarea nesupervizată. Acest tip de învățare este de obicei utilizat când datele sunt accesibile și etichetele sunt mai greu accesibile. Învățare semi-supervizată, de obicei, aplică o învățare nesupervizată unui set de date necunoscut, pentru a vedea cum sunt distribuite datele, iar apoi se aplică o învățare supervizată. [6] , [4]

2.1.2 Modul de învățare

Acesta se referă la modul în care algoritmul de învățare decide să ia o decizie și cum decide acesta să-și modifice sau actualizeze modelul de învățare. Există mai multe moduri diferite de învățare.

2.1.2.1 Modul de învățare Off-line

În acest mod, algoritmul de învățare ia o decizie după ce a parcurs toate exemplele, adică acesta o să ia o decizie după ce a analizat toate exemplele. Modificarea modelului este făcută la fel, după analiza tuturor exemplelor.

2.1.2.2 Modul de învățare On-line

În acest mod de învățare modelul este actualizat după prezentarea fiecărui exemplu. La apariția unui exemplu, algoritmul ia o decizie și își actualizează modelul pentru a putea învăța acel exemplu. [6]

2.1.3 Etapele procesului de învățare

Algoritmii de învățare sunt algoritmi care trebuie să ia decizii pe baza unei „experiențe”, aceasta fiind un model un model de învățare creat de algoritm pe baza informațiilor sau a datelor procesat de acesta în trecut. Pentru a se crea modelul de învățare, acești algoritmi trebuie sa parcurgă două etape : etapa de antrenare și etapa de testare.

2.1.3.1 Etapa de antrenare

În această etapă, pe baza exemplelor de antrenament, se face o analiză a exemplelor pentru a se obține o schemă de clasificare sau de clustering. Schema obținută reprezintă modelul de învățare. Aceasta este etapa în care algoritmul învață, etapa în care acesta își formează „experiența” pe baza căreia trebuie să ia o decizie în viitor.

2.1.3.2 Etapa de testare

În această etapă se încearcă o îmbunătățire a schemei create la etapa de antrenare printr-un proces de testare. Procesul de testare se face pe baza setului de date de test, acesta trebuie să fie diferit de cel de antrenament. Cu alte cuvinte se dorește o îmbunătățire a „experienței” algoritmului. [6]

2.2 Metode de normalizarea a datelor

Normalizarea se referă la rescalarea datelor de intrare din domeniul lor original într-un alt domeniu, un domeniu nou. Domeniul rezultat după aplicarea normalizării este de obicei mai îngust ([0,1], [-1,1], ...). [5]

Fiecare atribut este normalizat prin scalarea valorilor acestuia, astfel încât să se încadreze într-un interval mai mic. Scalarea atributelor se face pe baza valorilor inițiale ale atributelor.

Normalizarea se face în scopul îmbunătățiri preciziei algoritmilor de învățare automată și în scopul eliminării elementelor redundante din date. Aceasta oferă și o înțelegere mai bună a structurii în date de dimensiuni mari. [7]

Există multe metode de normalizare a datelor, iar în cele ce urmează scurtă prezentare a câtorva metodele de normalizare:

2.2.1 Normalizarea binară

Această metodă înlocuiește valoarea unui atribut cu „1" dacă înregistrare conține atributul sau cu „0" dacă înregistrarea nu conține atributul. Această metodă nu ține cont de valoare pe care o are acel atribut. Normalizarea binară este folosită atunci când se verifică dacă un exemplu conține sau nu un atribut, nu și când se ținem cont de frecventă de apariție a atributului în exemplu. [4]

2.2.2 Normalizarea nominală

Face trece valorii unui atribut în alt interval. Aceasta metodă face schimbarea de interval tinând cont de atributul cu valoarea cea mai mare din înregistrările acelui exemplu. Schimbarea de interval se face prin împărțirea valorii fiecărui atribut la valoarea maximă din înregistrările acelui exemplu. Formula care se aplică este următoarea:

$$v' = \frac{v}{MAX_{Atribut}} \tag{2.1}$$

unde:

- v' este valoarea atributului în intervalul [0,1], valoarea normalizată;
- v este valoarea inițială a atributului;
- $MAX_{Atribut}$ este valoarea atributului cu cea mai mare valoare din atributele acelui exemplu. [4]

2.2.3 Normalizarea Suma1

Această metodă de normalizare convertește valoarea fiecărui atribut din exemplu astfel încât la finalul normalizării suma atributelor din exemplu să fie egală cu „1. Schimbarea intervalului se face calculând suma valorilor tuturor atributelor din acel exemplu, iar apoi se împarte valoarea fiecărui atribut din exemplu la acea sumă. Formula care se aplică este următoarea:

$$v' = \frac{v}{\sum_{k=1}^{NrElemDoc} v_i} \tag{2.2}$$

unde:

- v' este valoarea nouă, valoarea normalizată;
- v este valoarea inițială, înainte de normalizare;
- $\sum_{k=1}^{NrElemDoc} v_i$ este suma tuturor valorilor atributelor din exemplul. [2]

2.2.4 Normalizarea min-max

Această metodă de normalizare face conversia dintr-un domeniu în altul făcând o transformare liniară de la datele de intrare. Pentru aplicarea acestei metode trebuie să se cunoască valorile minime și maxime ale atributelor. Este important și faptul că trebuie să se transmită formulei, să se cunoască, noile limite ale noului interval, noul minim și noul maxim. Această metodă presupune schimbarea domeniului de valori din [min, max] (ex: [0,10]) în [new_min, new_max] (ex: [0,1]) pe baza formulei [7] :

$$v' = \frac{v - min}{max - min}(new\_max - new\_min) + new\_min \tag{2.3}$$

unde:

- v' este noua valoare, valoarea normalizată din intervalul nou [new_min, new_max];
- v este valoarea inițială, valoarea înainte de normalizare care făcea parte din [min,max];
- min, max sunt limitele intervalului inițial;
- new_min, new_max sunt limitele noului interval.

2.2.5 Normalizarea Cornell-Smart

Aceasta este o metodă care face o normalizare în intervalul [0,2]. În intervalul [0,1] nu avem valori, dar apare și „0”, deoarece dacă valoarea atributului este „0” și data normalizată o să aibă valoarea „0”. Valorile normalizate, diferite de „0” o să facă parte din intervalul [1,2]. Normalizarea se face cu formula [4] :

$$v' = \begin{cases} 0, & dacă\ v = 0 \\ 1 + \log(1 + \log(v)), & altfel \end{cases} \quad (2.4)$$

2.3 Metode de calcul a similarității

Algoritmii de învățare automată încearcă să creeze modele de pe care le poate învăța. Crearea și actualizarea acestor modele se face pe baza asemănărilor dintre exemplele cu care lucrează algoritmii de învățare.

Similaritatea este o metodă prin care se poate determina gradul de asemănare dintre obiecte, iar disimilaritate poate determina cât de diferite sunt obiectele. [5]

Similaritatea sau disimilaritatea sunt reprezentate de un număr pozitiv care descrie cât de apropiate sau cât de depărtate sunt obiectele unul față de celălalt. Acestea se pot obține prin evaluări efectuate de către experți pe baza anumitor observații asupra obiectelor sau prin calcularea distanțelor dintre obiecte.

În învățarea automată trebuie să se calculeze distanța între fiecare două exemple pentru a calcula similaritatea sau disimilaritatea dintre exemple. [4]

Distanța este o valoare numerică care descrie cât de depărtate sau apropiate sunt două obiecte sau puncte unul față de celălalt. [8]

Distanța este definită în [1] astfel: Fie multimea nevidă M ($M \neq \emptyset$) . Se numește distanță pe M, orice aplicație $d: M \times M \rightarrow R$ cu următoarele proprietăți:

1. Nenegativitate:

   $d(x,y) \geq 0, \forall x,y \in M$

   Distanța între oricare două puncte nu trebuie să fie negativă.

2. Separare:

   $d(x,y) = 0 \Leftrightarrow x = y, \forall x,y \in M$

   Dacă distanța între două puncte este 0, atunci punctele sunt identice.

3. Simetria:

   $d(x,y) = 0 \Leftrightarrow x = y, \forall x,y \in M$

   Distanța între oricare două puncte este aceeași indiferent de direcție.

4. Inegalitatea triunghiului:

   $d(x,z) \leq d(x,y) + d(y,z), \forall x,y \in M$

   Distanța între oricare două puncte este distanța minimă în raport cu orice cale.

2.3.1 Tipuri de distanțe

Distanțele pot fi definite ca un număr real care arată cât de depărtate sunt două obiecte. [8] Există mai multe modalități de a calcula distanțele. În ceea ce urmează o să se descrie câteva modalități de calcul a distanțelor.

2.3.1.1 Distanța Manhattan

Aceasta este distanța între două puncte pe un drum parcurs pe axe perpendiculare, asemănătoare cu distanța parcursă în Manhattan între două puncte și ia valori în intervalul [0,∞) [3] . Numele se referă și la distanța parcursă de un taxi într-o rețea stradală pentru a ajunge din punctul original într-un anumit punct X. [10] Aceasta se calculează după formula:

$$d(p,q) = \sum_{i=1}^{n} |p_i - q_i| \tag{2.5}$$

2.3.1.2 Distanța Euclidiană

Aceasta este lungimea segmentului de dreaptă care unește două puncte dintr-un spațiu euclidian n-dimensional și are valorile în intervalul [0,∞) [4] . Această distanță se calculează cu formula:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{2.6}$$

2.3.1.3 Distanța Minkowski

Această distanță poate fi considerată o generalizare a celor două distanțe de mai sus, distanța euclidiană și distanța Manhattan. [9] Formula este:

$$d(p,q) = \left(\sum_{i=1}^{n} (|p_i - q_i|)^k \right)^{\frac{1}{k}} \tag{2.7}$$

Dacă k este 1, atunci se poate observa că se calculează distanța Manhattan, iar dacă k este 2 atunci calculează distanța euclidiană.

2.3.1.4 Distanța Cosinus

Aceasta este o distanță care măsoară disimilaritatea calculând unghiul dintre doi vectori. Distanța cosinus are valori în intervalul [0,1] și se calculează după formula [3] :

$$d(p,q) = \frac{\sqrt{\sum_{i=1}^{n} {p_i}^2} \times \sqrt{\sum_{i=1}^{n} {q_i}^2}}{\sum_{i=1}^{n} p_i \times q_i} \tag{2.8}$$

2.3.2 Matricea de similaritate

Este o matrice care memorează toate similaritățile dintre toate perechile de obiecte, aceasta memorează toate distanțele calculate între fiecare pereche de doua obiecte [3] . În majoritatea cazurilor aceasta este o matrice de n×n, unde n reprezintă numărul de obiecte sau exemple. Această matrice este de forma:

$$\begin{pmatrix} 0 & \cdot & \cdot & \cdot & \cdot \\ d(2,1) & 0 & \cdot & \cdot & \cdot \\ d(3,1) & d(3,2) & 0 & \cdot & \cdot \\ \vdots & \vdots & \vdots & \cdot & \cdot \\ d(n,1) & d(n,2) & \cdots & d(n,n-1) & 0 \end{pmatrix} \tag{2.9}$$

unde d(i, j) reprezintă similaritatea sau distanța dintre obiectul „i” și obiectul „j”. [5]

Datorită proprietăților distanței se poate observa că d(i, j) = d(j, i) și că pe diagonala principală sunt doar valori de 0, deoarece d(i, i) = 0.

Matricea de similaritate este des utilizată în învățarea automată, deoarece aceasta oferă algoritmilor de învățare automată toate similaritățile dintre exemple. În acest fel algoritmul are acces la toate similaritățile fără ca acesta să le mai calculez, astfel se scade timpul de execuție al algoritmilor.

2.4 Algoritmi de clustering

Clusteringul poate fi definit ca modalitatea prin care se face gruparea unor obiecte în funcție de anumite criterii sau anumite caracteristici de similaritate.

Un cluster este un grup de obiecte care sunt similare unul față de celălalt și sunt diferite de obiectele care nu fac parte din acel cluster. [4]

Clusteringul procesul de grupare a obiectelor în clase care conțin obiecte similare. Un cluster de date poate fi tratat ca un grup de date, deci poate să fie considerat ca un proces de compresie a datelor. În anumite aplicații acesta este cunoscut și ca segmentarea datelor, deoarece clusteringul împarte seturile mari de date în grupuri pe baza similarității.

În învățarea automată clusteringul este un exemplu de învățare nesupervizată, deoarece acesta este o formă de învățare care se bazează pe învățarea din observații, nu o formă de învățare care se bazează pe învățarea din exemple. [5]

2.4.1 Generalități

Algoritmii de clustering fac parte din categoria algoritmilor de învățare nesupervizată. Această categorie de algoritmi de învățare primesc la intrare doar setul de date, fără să se mai specifice și clasele din care fac parte exemplele din setul de date, iar aceștia trebuie să grupeze exemplele în clusteri cât mai relevanți.

Clusteringul trebuie să satisfacă două condiții foarte importante: că un cluster trebuie să conțină cel puțin un obiect și că obiectele nu au voie să facă parte din mai mult de un cluster, adică un obiect trebuie să fie inclus într-un singur cluster. [3]

Prin aplicarea algoritmilor de clustering se pot identifica regiunile dense sau separate din spțiul de obiecte și se pot descoperi modele sau corelații între atributele datelor. Acest tip de algoritmi este utilizat în numeroase aplicații cum ar fi: cercetarea pieței, marketing, analiza datelor cât și în procesarea de imagini.

2.4.1.1 Cerințele algoritmilor de clustering

Algoritmii de clustering trebuie să îndeplinescă anumite cerințe, în ceea ce urmează se vor prezenta câteva cerințe pe care acest tip de algoritmi trebuie să le îndeplinească conform cu [5] :

- **Scalabilitatea:** Se referă la posibilitatea de a lucra cu eșantioane din seturi de date mari. Algoritmii de clustering lucrează bine pe seturi de date mici, dar pe seturile de date mari pot exista rezultate neconvingătoare.
- **Abilitatea de a utiliza tipuri diferite de atribute:** Algoritmii de clustering sunt proiectați să lucreze cu date numerice, dar aplicarea acestora poate necesita clusterizarea altor tipuri de date, cum ar fi datele binare, nominale, ordinale sau o combinație între acestea.
- **Găsirea clusterilor de formă arbitrară:** Mulți algoritmi folosesc distanța Euclidiană sau Manhattan, iar acestea descoperă clusteri de formă sferică. Această cerință se se referă la importanța dezvoltării algoritmilor, astfel încât să poată detecta clusteri de forme diferite.
- **Determinarea parametrilor de intrare să necesite cunoașterea minimă a domeniului:** Parametrii de intrare ai algoritmului mai ales pe seturile mari de date, iar rezultatul aplicării algoritmului este determinat de aceștia. Această cerință se referă la determinarea parametrilor de intrare, astefel încât să putem controla, prin intermediul acestora, cât mai ușor calitatea clusteringului.

- **Abilitatea de a folosi date care conțin zgomot:** Se referă la abilitatea algoritmilor de a folosi date care conțin zgomot (date irelevante, neconcludente sau incomplete). Uni algoritmi de clustering sunt sensibili la zgomot și acest lucru poate duce la o calitate slabă a clusteringului.
- **Insensibilitatea la ordinea de prelucrare a datelor de intrare:** Există algoritmi de clustering care sunt sensibili la ordinea de prelucrare a datelor, adică oferă rezultate diferite în funcție de ordinea în care aceștia prelucrează datele. Este important să se dezvolte algoritmi care să nu fie sensibili la ordinea de prelucrare a datelor.
- **Dimensionalitate mare:** Datele cu care se lucrează pot avea mai multe dimensiuni sau atribute. Algoritmii de clustering lucrează bine cu date de dimensiuni mici, care au doar două sau trei dimensiuni. Algoritmii care lucrează cu date care au un număr mare de dimensiuni sunt greu de găsit, deoarece datele respective sunt greu de separat.
- **Clustering bazat pe anumite constrângeri:** În aplicațiile de clustering care interacționează cu lumea reală trebuie să se țină cont de anumite constrângeri. O provocare este de a se găsi grupuri, prin clustering, care să satisfacă anumite constrângeri specificate.
- **Interpretabilitatea și utilizarea:** Rezultatele care apar după aplicarea algoritmilor de clustering trebuie să fie interpretabile, ușor de înțeles și reutilizabile. Este important de știut că scopul aplicației poate să influențeze clusteringul.

Cu aceste cerințe se poate face o analiză asupra procesului de clustering. Se poate analiza dacă rezultatele clusteringului au fost influențate de aceste cerințe și care sunt tipurile de date care pot fi clusterizate. Pe baza acestora se poate alege un algoritm cât mai specific pentru scopul aplicației care a implementat algoritmul de clustering.

2.4.1.2 Clasificarea metodelor de clustering

Crearea grupurilor din date se poate face folosind diverse metode și diverse strategii. Este dificil să se facă o clasificare clară a metodelor de clustering, deoarece o metodă poate aveam caracteristici din mai multe categorii. În general, metodele de clustering pot fi clasificați în următoarele categorii, conform cu [5] .

Metode bazate pe partiționarea datelor: Considerând că există o baza de date cu „n” elemente și că acestea trebuie grupate în „k” grupuri, acest tip de algoritmi construiește „k” patriții ale datelor unde fiecare partiție reprezintă un cluster și $k \leq n$. O astfel de metodă creează o partiționare inițială, iar apoi pe baza anumitor criterii se încearcă îmbunătățirea partiționării prin mutarea elementelor dintr-un grup în altul. Acest tip de metode funcționează bine pentru gruparea datelor în formă sferică din baze de date mici și mijloci.

Metode ierarhice: Această categorie de metode realizează o structură ierarhică a setului de date. Acestă categorie poate avea două abordări, aglomerativă sau divizivă, depinde de cum este formată structura ierarhică.

- aglomerativă: aceasta are o abordare „bottom-up”, în care la început fiecare obiect este considerat ca fiind un cluster, iar apoi clusteri se unesc pas cu pas pe baza unor caracteristici de similaritate până la îndeplinirea unei condiții de oprire dată sau până când toate obiectele formează un sigur cluster.
- divizivă: aceasta are o abordare „top-down”, în care la început toate obiectele sunt considerate în același cluster, iar apoi se divide fiecare cluster în clusteri mai mici până când la îndeplinirea unei condiții de oprire dată sau până când fiecare obiect formează un cluster.

Această categorie are o problemă care constă în faptul ca odată ce pasul de unire sau divizare este efectuat acesta nu mai poate fi anulat. [4]

Metode bazate pe densitate: Ideea generala al acestor metode este de a se continua creșterea clusterului atâta timp cât numărul obiectelor din vecinătate (densitatea) depășește un anumit prag, adică pentru fiecare obiect dintr-un cluster, vecinătatea unei raze date trebuie să conțină cel puțin un număr minim de obiecte. Aceste metode sunt folosite pentru a detecta și elimina zgomotul din date, cât și pentru a forma grupuri de forme arbitrare.

Metode bazate pe rețele: Aceste metode cuantifică spațiul obiectului într-un număr finit de celule care formează o structură de rețea, iar toate operațiile de grupare sunt efectuate pe structura rețelei, adică pe spațiul cuantizat.Avantajul principal al acestei metode este timpul rapid de procesare, deoarece acesta depinde numai de numărul de celule din fiecare dimensiune din spațiul cuantizat, nu de numărul de obiecte din date, fiind independent de acesta.

Metode bazate pe modele: Acestea se bazează pe crearea unui model pentru fiecare cluster și caută cea mai bună potrivire a datelor la modelul dat. Un algoritm care folosește acest model detectează grupurile prin construirea unei funcții de densitate care reflectă distribuția spațială a obiectelor din date. Acest model detectează automat numărul de clusteri pe baza unor statistici. Folosind acest model se poate detecta zgomotul sau valorile aberante și produce metode de clusterizare robuste.

Alegerea algoritmului de clustering depinde atât de tipul de date pe care se aplică algoritmul, cât și de scopul aplicației care implementează algoritmul. Există algoritmi de clustering care folosesc principiile mai multor metode prezentate anterior, astfel încât este dificil să se considere că un algoritm aparține în mod special doar unei categorii de metode de clustering, iar în anumite aplicații pot avea nevoie implementarea mai multor metode de clustering. [5]

2.4.2 Algoritmul DBSCAN

Acesta este un algoritm de clustering care este bazat pe densitate și ne oferă posibilitatea de a detecta „zgomotul” sau elementele redundante din date.

DBSCAN (**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise) [2] este un algoritm care estimează densitatea prin contorizarea numărului de obiecte dintr-o arie care are o rază fixă. Acesta consideră două obiecte ca fiind conectate dacă acestea se află unul în vecinătatea celuilalt.

Algoritmul DBSCAN crește regiuni cu densitate suficient de mare în clusteri. Acesta descoperă clusteri de forme arbitrare în baze de date spațiale care conțin zgomot. Un cluster este definit ca un set maxim de obiecte conectate la o densitate. [5]

Prezentarea câtorva definiții care ajută la o înțelegere mai bună ale algoritmilor de clustering bazați pe densitate.

- Vecinătatea de o anumită rază **ε** numește **vecinătate ε**. [5]
- Un obiect este numit obiect central (**core object**) dacă în vecinătate ε a acestuia există un număr minim de obiecte (**minObj**) care formează densitatea, adică densitatea din vecinătatea acestuia trebuie să depășească o anumită valoare de prag. [2]
- Un obiect *p* este direct conectat cu un obiect *q* dacă *p* este în vecinătatea ε a lui *q* și *q* este un obiect central.
- Un obiect *p* este **conectat la densitate** (**density-connected**) cu un obiect *q* dacă exită un obiect *o* care este direct conectat atât la *p* cât și la *q* respectând vecinătatea ε și numărul minim de obiecte (minObj) . [5]
- Un cluster este un set de obiecte conectate la densitate, care are un număr de obiecte mai mare decât numărul minim de obiecte care pot forma o densitate.
- Zgomotul este definit ca setul de obiecte care nu fac parte din nici un cluster.

Scopul principal al algoritmului DBSCAN este acela de a creea toate clusterele care respectă raza minimă a ariei de vecinătate (vecinătate ε) și numărul minim de obiecte care formează o densitate (minObj).

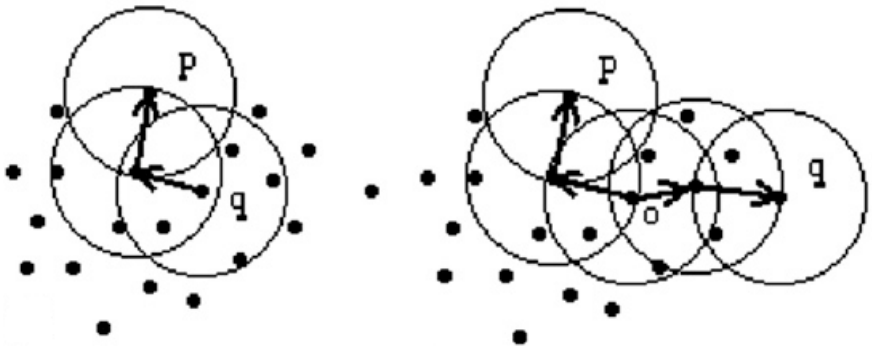


Fig. 2.1 Crearea densităților în DBSCAN

În DBSCAN există trei tipuri diferite de obiecte:

- obiecte centrale (**core objects**): sunt obiectele care au o vecinătate densă,
- obiecte de margine (**border objects**): sunt obiectele care aparțin clusterului, dar nu au o vecinătate densă.
- obiecte de zgomot (**noise objects**): obiectele care nu fac parte din nici un cluster.

Fig. 2.2 Tipurile de obiecte în DBSCAN

DBSCAN are o proprietate importante care ajută la un calculul mai eficient. Aceasta se referă la faptul că un cluster care respectă raza minimă a ariei de vecinătate și numărul minim de puncte care formează o densitate este determinat în mod unic de oricare dintre obiectele sale centrale.

Complexitatea algoritmului DBSCAN în cel mai rău caz este O($n \log n$), unde n este numărul de obiecte din setul de date. Din cauză că acesta este face o indexare spațială se poate ajunge la o complexitate de O(n^2) când se lucrează cu date care au multe dimensiuni. [2]

Pașii pe care algoritmul îi parcurge ar pute fi [5] :

1. Se ia un obiect aleatoriu din baza de date și se verifică dacă a fost sau nu analizat, dacă nu a fost se trece la pasul 2.
2. Se calculeaza distanță între obiectul ales și celelalte obiect deja analizate și se decide sau nu formarea/actualizarea unui cluster.

3. Se verifică dacă obiectele formează un grup (cluster) pe baza numărului minim de obiecte care pot forma o densitate, dacă numărul de obiecte este mai mare decât numărului minim de obiecte care pot forma o densitate se formează un cluster, dacă nu exemplele sunt grupate ca zgomot.
4. Algoritmul se repetă până când toate obiectele au fost analizate.

Algoritmul poate fi reprezentat în pseudocod astfel:

Notații:

- **X** = {$x_1 \ldots x_n$} (datele care trebuie să fie clusterizate)
- **eps** (distanța minimă între 2 puncte)
- **minPts** (numărul minim de pucte care poate forma un cluster)
- **distFunc** (o functie pentru calculul distanțelor)
- **Cluster** (contorul clsterului)

Fig. 2.3 Notații folosite în pseudocod

```
/* verifică pe baza lui eps distanța între K și toate datele din X*/
/*și întoarece o zona care are distanța intre puncte mai mică ca eps  */
CreazaRegiune(X, distFunc, k, eps){
        L := lista vidă         /*lista o să conțină regiunea*/
        for each d  in X {
                if (distFunc(k,d) < eps)
                {
                        L := L ∪ {d}          /*daca distanța este mai mică ca eps se introduce d in listă*/
                }
        }
        return L
}
```

Fig. 2.4 Funcția care creează o regiune pe baza vecinătăți ε

```
DBSCAN( X, distFunc, eps, minPts ){
    Cluster := 0                /* initializăm contorul clusterului cu 0 */
        for each d in X {               /*se parcurg toate datele*/
                if (eticheta(d) = nedefinită ) {                /*se verifică dacă a fost sau nu analizat*/
                        V := CreazaRegiune(X, distFunc, d, eps)         /*regiunea din care aparține d*/
                        if (length(V) < minPts){
                                eticheta(d) = zgomot     /*dacă nu se indeplinesc numărul minim de date*/
                                                        /*atunci data d este etichetată ca zgomot*/
                                next                    /*trecem la următoarea iterație*/
                        }
                        Cluster := Cluster + 1           /*se trece la următoare etichetă a clusterului*/
                        eticheta(d) := Cluster          /*se eticheteză data cu cluster următor*/
                        G := V \ {d}                            /*se iau elementele din regiune fără d*/
                        for each q in G {
                                if ( eticheta(q) = zgomot) {eticheta(q) := Cluster} /*se schimbă din zgomot în
                                                                                        date de margine*/

                                if (eticheta(d) = nedefinită)
                                {
                                        eticheta(q) := Cluster
                                        V := CreazaRegiune(X, distFunc, d, eps)
                                        if (length(V) ≥ minPts){
                                                G := G ∪ V
                                        }
                                }
                        }
                }
        }
}
```

Fig. 2.5 Funcția care implementează algoritmul DBSCAN

Algoritmului DBSCAN are următoarele avantaje:

- Nu necesită numărul de clusteri care trebuie să rezulte după aplicare, spre deosebire de alți algoritmi de clustering.
- Acesta poate să grupeze date în formă arbitrară.
- Poate folosi date care conțin zgomot.
- Parametrii de intrare necesită o cunoaștere minimă a domeniului.
- Nu este sensibil la ordinea de prelucrare a datelor.

Acest algoritm are totuși și anumite dezavantaje:

- Dacă datele cu care lucrează au o dimensiune mare, crește mult complexitatea, ajungând la o complexitate de O(n^2) la rulare, unde n este numărul de obiecte din baza de date.
- Poate trata punctele de frontieră ca fiind zgomot.
- Nu grupează bine seturile de date cu diferențe mari de densitate.

2.4.3 Algoritmul HAC (Hierarchical agglomerative clustering)

Acesta este un algoritm de tip ierarhic care folosește o abordare „bottom-up”, adică acesta creează o structură ierarhică a setului de date folosind o abordare aglomerativă.

Algoritmul HAC, consideră că fiecare obiect este un cluster diferit și încearcă să unească clusterii mai mari până când toate obiectele formează un cluster sau până când este îndeplinită o condiție de oprire. Unirea clusterilor se face pe baza similarității dintre aceștia. [5]

Algoritmii ierarhici au fost dezvoltați pentru a se rezolva anumite constrângeri pe care le au algoritmii bazați pe partiționarea datelor, deoarece aceștia folosesc un mecanism mult mai flexibil pentru clusterizarea obiectelor din date.

Un beneficiu adus de algoritmii ierarhici față de cei de partiționare a datelor este acela că nu trebuie să se specifice numărul de clusteri care trebuie să rezulte la final. Acesta permite oprirea dezvoltării ierarhiei la orice nivel de corespondență prin stabilirea unui prag. Ierarhia clusterelor se numește **dendrogramă.** [2]

Dendrogramă este o diagrama care descrie cum s-au format, respectiv unit clusteri în gruparea ierarhică. [12]

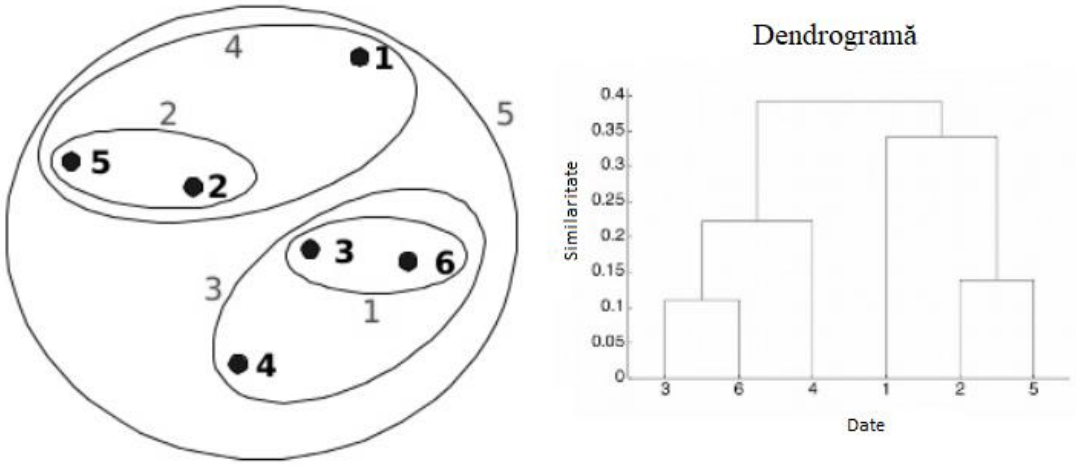


Fig. 2.6 Exemplu de dendrogramă pentru algoritmul HAC

Pașii pe care algoritmul îi parcurge ar pute fi [4]:

1. Se numerotează toat obiectele de la 1 la n, unde n este numărul total de documente. Fiecare document se consideră un cluster.
2. Se calculează matricea de similaritate, care o să conțină distanța între fiecare obiect și toate celelalte obiecte.
3. Se determină clusterii care sunt cei mai similari, adică cea mai mică distanță din matricea de similaritate.
4. Se unesc cei doi clusteri într-un cluster nou și dacă distanța este mai mare decât un anumit prag se termină algoritmul, dacă nu se calculează distanțele între clusterul nou și toți ceilalți clusteri. Dacă nu s-a terminat algoritmul trebuie să se șteargă coloanele și rândurile care corespund celor doi clusteri uniți, din matricea de similaritate și să se adaugă o coloană și un rând care să corespundă noului cluster creat.
5. Algoritmul se repetă până când rămâne un singur cluster sau până când este îndeplinită condiția de la pasul 4.

Algoritmului HAC are următoarele avantaje:

- Nu trebuie specificat numărul de clusteri rezultat.
- Parametrii primiți de acesta la intrare necesită o cunoaștere minimă a domeniului.
- Nu este sensibil la ordinea de prelucrare a datelor.

Algoritmului HAC are totuși și anumite dezavantaje:

- Complexitate este O(n^2), unde n este numărul de obiecte care trebuie grupate.
- Este sensibil la zgomot și la valorile redundante

Algoritmul HAC unește clusteri pe baza similarității acestora. Similaritatea dintre clusteri este exprimată prin distanța dintre doi clusteri. [4]

La începutul algoritmului fiecare cluster este reprezentat de un obiect, deci similaritatea clusterilor poate să fie reprezentată de matricea de similaritate, dar pe parcurs clusteri se unesc si trebuie să se determine un nou mod de a calcula similaritatea între clusteri. Există mai multe metode de a calcula similaritatea între clusteri [4] , [2]:

2.4.3.1 Single link

În această metodă, similaritatea între doi clusteri este dată de similaritatea obiectelor care sunt cele mai similare din cei doi clusteri. Cu alte cuvinte similaritatea este dată de distanța minimă între două obiecte care fac parte din cei doi clusteri, adică cei mai apropiați vecini din cei doi clusteri. Această metodă oferă mai multă importanță regiunilor în care clusteri sunt mai apropiați, deci această metodă se bazează pe similarități locale. Datorită similarității locale, metoda grupează bine datele care nu sunt oval, dar un dezavantaj este ca devine sensibil la zgomot sau la datele cu valori aberante.

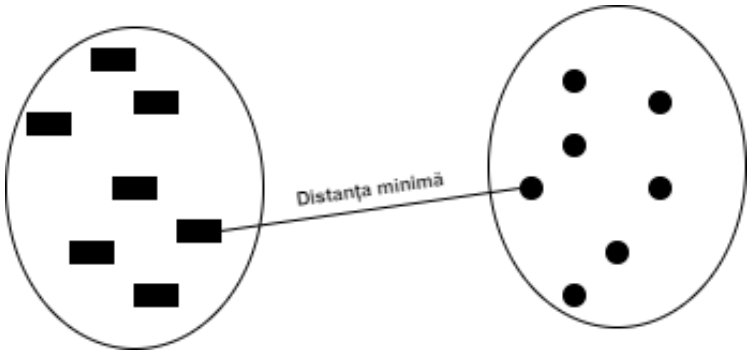


Fig. 2.7 Single link

$$d_{SingleLink}(X, Y) = min_{x \in X, y \in Y} d(x, y) \tag{2.10}$$

2.4.3.2 Complete link

În această metodă, similaritatea dintre doi clusteri este dată de similaritatea obiectelor care sunt cele mai disimilare din cei doi clusteri. Similaritatea este dată de distanța maximă dintre două obiecte care fac parte din cei doi clusteri, adică cei mai depărtați vecini din cei doi clusteri. Metoda complete link nu are un comportament local și se obțin, de obicei, clusteri compacți, deoarece această metodă ia în considerare structura clusterului. Această metodă este sensibilă la zgomot și la valori aberante, la fel ca în cazul metodei single link.

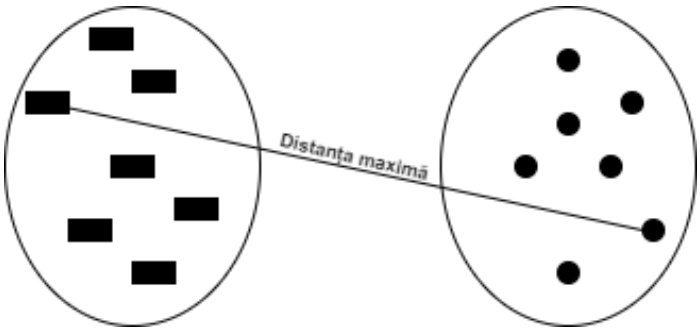


Fig. 2.8 Complete link

$$d_{CompleteLink}(X,Y) = max_{x\in X, y\in Y} d(x,y) \tag{2.11}$$

2.4.3.3 Average link

Această metodă ia în considerare similaritatea între toate perechile de obiecte din cei doi clusteri, astfel se diminuează dezavantajele de la single link și complete link. Această metodă consideră similaritatea dintre doi clusteri ca fiind media similarităților obiectelor din cei doi clusteri, adică se face suma tuturor distanțelor dintre cei doi clusteri și se împarte la numărul elementelor din cei doi clusteri.

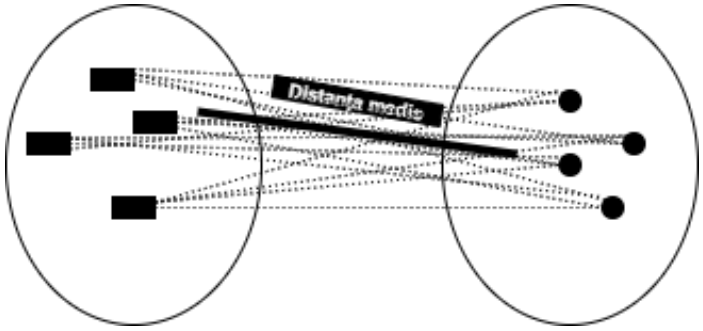


Fig. 2.9 Average link

$$d_{AverageLink}(X,Y) = \frac{1}{|X||Y|}\sum_{x\in X, y\in Y} d(x,y) \tag{2.12}$$

2.4.3.4 Centroid link

Cu această metodă, similaritatea dintre clustere este calculata ca fiind similaritatea dintre centroizi celor doi clusteri. Această metodă calculează distanța dintre centroizi celor doi clusteri.

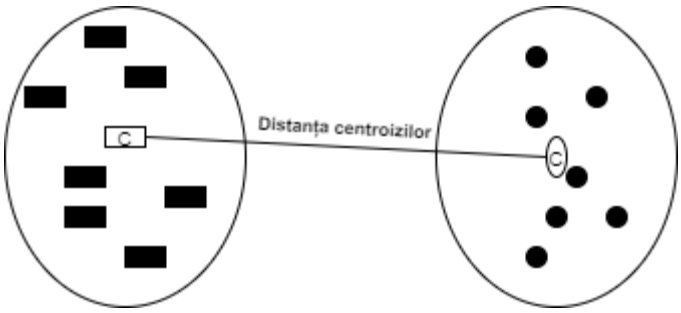


Fig. 2.10 Centroid link

$$d_{CentroidLink}(X, Y) = \|C_X - C_Y\| \tag{2.13}$$

2.5 Metode de evaluare

Algoritmii diferiți de clustering pot oferii rezultate diferite pe același set de date. Rezultatele pot fi influențate de metoda pe care se bazează algoritmul, de capacitatea lui de a putea fi aplicat pe seturi care conțin date redundante sau chiar de parametrii de intrare de care acesta are nevoie. Este posibil ca același algoritm de clustering să ofere rezultate diferite chiar dacă este aplicat pe același set de date, deoarece rezultatele sunt influențate și de metoda folosită pentru calculul similarității. [3]

Pentru a se determina care algoritm este mai potrivit pentru problema la care este folosit sau pentru a determina ce metrică de similaritate trebuie să fie folosită de către algoritm este necesar să se facă o evaluare a rezultatelor oferite de către algoritm. Pentru a se evalua un algoritm trebuie să evaluăm performanța sau calitatea algoritmilor. Performanța sau calitatea algoritmilor trebuie să fie stabilite pe baza unor măsuri obiective. Există două tipuri de măsuri care pot fi folosite pentru evaluarea performanței sau a calității algoritmilor de învățare automată:

- **Măsuri externe:** este atunci când există o cunoaștere apriorie despre setul de date, adică atunci când datele sunt pre-etichetate.
- **Măsuri interne:** este atunci când nu avem o cunoaștere apriorie despre setul de date, când datele nu au etichete sau nu există informații despre rezultatele la care trebuie să ajungem.

Pentru algoritmii de clasificare se pot aplica doar măsurile externe, în timp ce pentru algoritmii de clustering putem aplica atât măsurile externe cât și pe cele interne. Se poate face o analiză a acestor măsuri și pe baza acelei analize să se determine dacă un algoritm oferă rezultate bune sau nu. [4]

2.5.1 Măsuri externe de evaluare

Aceste măsuri ajută la analiza rezultatelor oferite de către algoritmi, iar pe baza acestor analize se poate face o evaluare a performanței sau a calității algoritmului respectiv. Măsurile externe pot fi aplicate dacă există o cunoaștere apriorie despre setul de date, adică dacă înainte de aplicarea algoritmilor există informații despre rezultatele la care trebuie să ajungă algoritmul. Aceste măsuri pot fi aplicate doar dacă seturile de date din faza de testare sunt pre-etichetate. [4]

Algoritmii de clustering grupează datele din mai multe perspective, în funcție de metoda pe care este bazat fiecare algoritm. Datorită acestui motiv algoritmii de clustering ingnoră categoriile sau etichetele datelor până la sfârșitul fazei de antrenare. La sfârșitul acestei faze se creează o corespondență între clusteri și clase. Fiecare cluster primește o etichetă pe baza numărului etichete din acel cluster, adică fiecare cluster primesc eticheta datelor care apar de cele mai multe ori în cluster și au aceea etichetă. În faza de testare trebuie să se creeze matricea de eroare (**confusion matrix**), iar cu ajutorul acesteia se pot calcula măsurile externe de evaluare.

Matricea de eroare [11] este un tabel prin care se poate determina cât de bine a fost prezisă o etichetă de la intrare de către algoritmul de învățare automată. Această matrice trebuie făcută separat pentru fiecare etichetă și este important de specificat că acestea trebuie să fie actualizate simultan.

Etichetă	*Etichetă reală*		
Etichetă predicționată		Pozitiv	Negativ
	Pozitiv	**TP**	**FP**
	Negativ	**FN**	**TN**

Fig. 2.11 Matricea de eroare pentru un exemplu

Matricea de eroare este o matrice de 2×2 care conține următoarele câmpuri:

- **TP** (**T**rue **P**ositive): este un număr care reprezintă de câte ori această etichetă a fost predicționată și trebuia să fie predicționată. True Positive arată de câte ori eticheta corectă a fost prezisă ca fiind corect.
- **FP** (**F**alse **P**ositive): este un număr care reprezintă de câte ori această etichetă a fost predicționată și nu trebuia să fie predicționată. False Positive arată de câte ori eticheta greșită a fost predicționată ca fiind corectă.

- **FN** (**F**alse **N**egative): este un număr care reprezintă de câte ori această etichetă nu a fost predicționată și trebuia să fie predicționată. False Negative arată de câte ori eticheta corectă a fost predicționată ca fiind greșită.
- **TN** (**T**rue **N**egative): este un număr care reprezintă de câte ori această etichetă nu a fost predicționată și nu trebuia să fie predicționată. True Negative arată de câte ori eticheta greșită a fost predicționată ca fiind greșită.

Cu ajutorul matricei de eroare se pot calcula mai multe măsuri externe, dar în continuare sunt prezentate doar trei dintre acestea. [11] , [4]

- **Acuratețea:** această măsură reprezintă procentul de obiecte care au primit eticheta corectă din numărul total de obiecte care trebuiau să primească eticheta. Aceasta arată procentul obiectelor grupate corect de către algoritm. Formula folosită este următoarea:

$$Acuratețea = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{2.14}$$

- **Precizia:** această măsură reprezintă probabilitatea ca un obiect grupat corect să fie și util. Acesta arată procentul obiectelor grupate corect din totalul de obiecte pe care algoritmul a decis să le grupeze corect. Precizia se calculează cu următoarea formulă:

$$Precizia = \frac{(TP)}{(TP + FP)} \tag{2.15}$$

- **Recall:** această măsură reprezintă probabilitatea ca un obiect util să fie și grupat corect. Aceasta arată procentul obiectelor corecte din totalul de obiecte care trebuiau grupate corect. Formula folosită pentru a calcula această măsură este următoarea:

$$Recall = \frac{(TP)}{(TP + FN)} \tag{2.16}$$

Este foarte important de specificat faptul ca aceste măsuri sunt calculate pe fiecare matrice de eroare în parte, adică pentru fiecare etichetă separat. De obicei după ce acestea sunt calculate pentru toate etichetele se face o medie aritmetică a fiecărei măsuri, această medie reprezentând măsura totală, de exemplu se face o medie aritmetică a tuturor măsurilor de acurateț̦e și această medie reprezentând acuratețea totală.

2.5.2 Măsuri interne de evaluare

Aceste măsuri se aplică doar algoritmilor de clustering. Există multe tipuri de astfel de metrici, dar sunt prezentate doar două [3] :

1. **Compactness:** acesta măsoară cât de compactate sunt datele în clusteri, adică cât de apropiate sunt obiectele care fac parte din același cluster. Aceasta este reprezentată prin suma tuturor distanțelor la pătrat între centroidul clusterului și toate elementele care aparțin acelui cluster.

$$compactness(c) = \sum_{i=1}^{N} (c - x_i) \tag{2.17}$$

unde:

- N este numărul de elemente din cluster;
- c centroidul clusterului;
- x_i elementul i din cluster.

2. **Separability:** această metrică măsoară diferența dintre clusterii creați, adică pentru fiecare cluster se determină care este cel mai aproiat cluster. Aceasta calculează suma distanțelor dintre centroizii tuturor clusterelor.

$$separability(c) = \sum_{i,j=1}^{N_c} (c_i - c_j) \tag{2.18}$$

unde:

- N_c este numărul de clusteri formați;
- c_i centroidul clusterului i;
- c_j centroidul clusterului j.

3 Considerații practice

3.1 Structura aplicației realizate

Algoritmul DBSCAN și HAC sunt doi algoritmi care se bazează pe două metode diferite, unul este bazat pe metode ierarhice și unul pe metode bazate pe densitate. Algoritmii de clustering depind și de forma datelor de intrare, de parametrii cu care aceștia lucrează cât și de metrica de similaritate pe care o folosesc. Pentru a se putea determina performanța sau calitatea algoritmului în gruparea punctelor în spațiul 2D sau a documentelor din baza de date Reuters, am creat o aplicație care creează un proces prin care se face o analiză metrică a algoritmilor. Pentru a vedea eficiența unui algoritm de clustering trebuie să ținem cont de toți factorii prezentați anterior, de aceea procesul conține șase etape, primele etape având diferite metode de calcul sau de implementare, iar ultimele fac diverse calcule care ajută la calculul metricilor cu ajutorul putem face o evaluare a algoritmului, respectiv al procesului.

Primele etape au diferite metode de calcul sau de implementare, deoarece dorim să facem o analiză a algoritmilor pe diferite tipuri de fișiere, diferite forme de reprezentare a datelor și pe diferite metrici de similaritate. Se fac mai multe metode pentru fiecare etapă pentru că fiecare etapă influențează algoritmul de clustering și dorim să vedem care metodă din etapă influențează cel mai bine algoritmul. Ultimele etape implementează metode pe baza cărora se face o corespondență între etichetele originale ale datelor și grupurile create. Pe baza acestor corespondențe se determină dacă datele au fost grupate corect sau nu prin calcularea metricilor externe.

Evaluarea algoritmului trebuie făcută aplicând diferite metode la fiecare etapă și pe baza metricilor de la final să facem o analiză, astfel încât să determinăm care metodă din fiecare etapă este mai eficientă pentru fiecare algoritm. După, se analizează rezultatele fiecărui algoritm și se face o evaluare a eficienței algoritmului.

Prin intermediul acestui proces nu se evaluează doar dacă algoritmul a grupat corect sau nu datele. Putem evalua și anumite cerințe pe care algoritmul trebuie să le îndeplinească, cu ar fi scalabilitatea, abilitatea de a folosi date care conțin zgomot, găsirea clusterilor de formă arbitrară, insensibilitatea la ordinea de prelucrare a datelor, etc. Acest proces ne ajută să facem și o evaluare a algoritmilor pentru a determina ce constrângeri au aceștia și pentru a determina care sunt avantajele și dezavantajele fiecărui algoritm, cât și în ce context este eficient algoritmul.

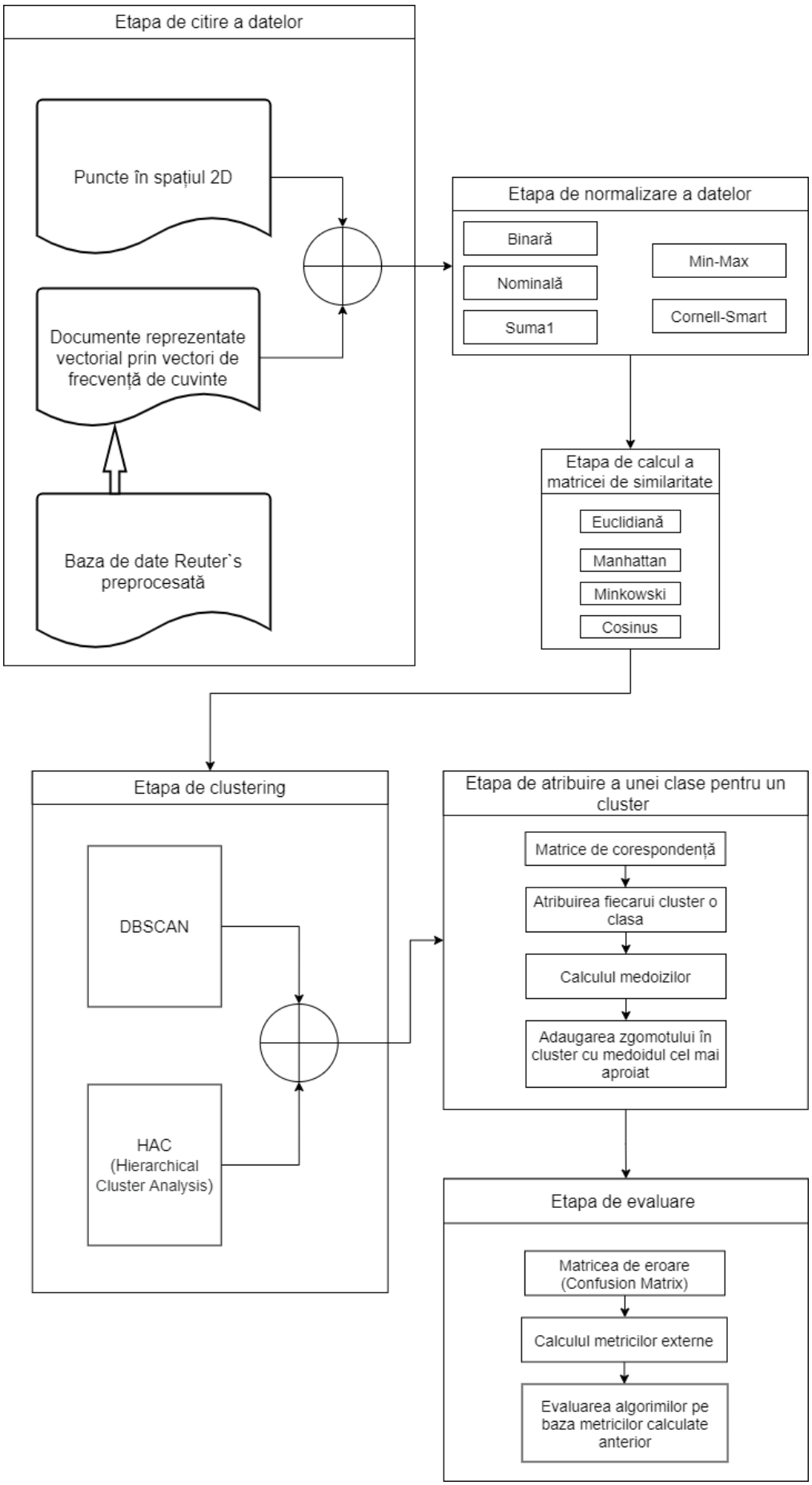


Fig. 3.1 Structura aplicației

În Fig. 3.1 este o schemă care reprezintă structura aplicației. Această schemă arată cele șase etape ale procesului și ce face fiecare etapă. Schema arată și metodele de calcul sau de implementare folosite de fiecare etapă. Fiecare etapă din acest proces are un rol important și ajută la analiza performanței algoritmilor de clustering.

- **Etapa de citire a datelor:** această etapă face citirea datelor din diferite tipuri de fișiere, în care datele sunt sub diferite. Un tip conține puncte în spațiul 2D, iar celălalt documente care conțin vectori de tipul „*atribut:frecvență”*. Această etapă creează și structura de reprezentare a datelor cu care lucrează algoritmul.
- **Etapa de normalizare a datelor:** această etapă face trecerea, structurii de date creată de etapa de evaluare, din domeniul original într-un alt domeniu, un domeniu mai restrâns. Această etapă implementează mai multe metode prin care se face normalizarea datelor.
- **Etapa de a matricei de similaritate:** în această etapă se calculează similaritatea dintre fiecare exemplu, cu care lucrează algoritmul, cu toate celelalte exemple. Similaritatea este reprezentată de distanța dintre obiecte. La finalul acestei etape rezultă o matrice care o să conțină distanța între fiecare exemplu cu toate celelalte exemple. Această etapă implementează mai multe metode folosite la calculul distanțelor.
- **Etapa de clustering:** în această etapă se alege un algoritm de clustering, se introduc parametri necesari și se aplică algoritmul. La finalul acestei etape o sa avem o grupare a datelor. Aici sunt implementați cei doi algoritmi și amândoi se folosesc de matricea de similaritate pentru a grupa datele.
- **Etapa de atribuire a unei clase pentru un cluster:** această etapă creează o corespondență între fiecare cluster rezultat după aplicarea algoritmului și etichetele datelor. Se calculează elementul cel mai reprezentativ din fiecare cluster și cu ajutorul acestuia atribuim fiecărui exemplu o etichetă.
- **Etapa de evaluare:** în această etapă se creează matricea de eroare (confusion matrix) și pe baza acesteia se calculează acuratețea, precizia și recall-ul. În această etapă se verifică dacă algoritmul a grupat corect exemplele verificând eticheta inițială și eticheta după aplicarea algoritmului.

La finalul procesului se afișează un tabel rezultatele metricilor pe fiecare etichetă și media aritmetică a acestora. În tabel sunt și valorile parametrilor folosiți de acel algoritm. Pe baza rezultatelor din tabel se poate face o analiza asupra algoritmului și se pot compara rezultatele obținute de cei doi algoritmi.

3.2 Etapa de citire a datelor

Prima etapa pe care doresc să o descriu este etapă de citire a datelor. În această etapă se citesc datele cu care lucrează procesul de clustering. Această etapă se referă la metoda de reprezentare a datelor din fișiere (Matrice, Vector, Lista, Dicționare, etc) și la tipul de date pe care îl folosim pentru reprezentarea datelor. Etapa de citire a datelor este importantă pentru procesul de clustering, deoarece cu cât datele sunt mai bine reprezentate și tipul de date folosit este de o dimensiune optimă, atunci și procesul de clustering o să se execute într-un timp cât mai optim și o să ocupe mai puțină memorie.

Etapa de citire a datelor, chiar dacă pare o etapă minimală sau una care nu necesită interes, are nevoie de o atenție mai sporită, deoarece timpul de execuție a procesului de clustering, performanța procesului și memoria de care procesul are nevoie, depind într-o mare măsură de această etapă. De exemplu, la prima încercare de implementare a acestei etape am folosit o colecție generică numită „Dictionary<TKey, TValue>”. Această colecție este foarte ușor de folosit și foarte ușor de implementat, dar o problemă a acestei colecții este memoria mare pe care aceasta o ocupă și complexitatea ridicată de accesare a elementelor care aparțin acestei colecții. Ca alternativă am schimbat modul de reprezentare a datelor într-o matrice, pentru a reduce complexitatea de accesare a unui element. Am reușit să reducem memoria pe care o ocupă datele cu care lucrează procesul de clustering. Această reducere a fost posibilă datorită unei analize asupra domeniului de valori din care fac parte datele, iar datorită acestei analize s-a demonstrat că nu avem nevoie de mai mult de tipul de date Int16 pentru a reprezenta toate datele. La prima încercare de implementare, datorită colecției generice „Dictionary<TKey, TValue>”, datele ocupau 32MB, iar când am făcut o matrice de tipul Int16, datele ocupau 14MB. Se poate observa că, datorită modului și tipului de date folosit în reprezentarea datelor, am reușit să facem o optimizare de memorie de 56,25%, adică de 18MB.

Structura datelor de intrare pentru acest proces de clustering este de forma unei matrici de date. Aceasta este o matrice de **n × m**, unde „**n**” resprezintă numarul de exemple sau de documente pe care dorim să le grupăm și „**m**” reprezintă numărul de atribute sau caracteristici al fiecărui exemplu pe care dorim să îl grupăm.

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1(m-1)} & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2(m-1)} & x_{2m} \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ x_{(n-1)1} & x_{(n-1)2} & \cdots & x_{(n-1)(m-1)} & x_{(n-1)m} \\ x_{n1} & x_{n2} & \cdots & x_{n(m-1)} & x_{nm} \end{pmatrix} \quad (3.1)$$

Datele pot avea o mulțime diferită de atribute și de dimensiuni, adică exemplele de antrenament sunt de dimensiuni diferite. În caz că se aplică algoritmul de clustering asupra datelor care reprezintă puncte pe un grafic, acesta este un spațiu cu două dimensiuni, unde exemplele o să fie de dimensiunea doi. Dimensiunea este dată de numărul de atribute, în acest caz numărul de atribute este doi, deoarece exemplul este descris de valoarea punctului pe axa X și valoarea punctului pe axa Y (exemplu={x,y}). În cazul în care aplicăm algoritmul asupra datelor care reprezintă documente, vectori care conțin frecvența de apariție a cuvintelor în documente, atunci avem un spațiu cu N dimensiuni, unde numărul de atribute care descriu un document este N, iar prin urmare exemplele de antrenament o să fie de dimensiune N.

Am ales această structură de reprezentare a datelor din dorința de a satisface ideea de „Dimensionalitate”. Această idee se referă la problema prezentată anterior, referitor la numărul diferit de dimensiuni pe care le pot avea exemplele pe care dorim să le grupăm. Dimensionalitatea se referă la cât de ușor se poate adapta procesul de clustering, astfel încât acesta să poată lucra cu exemple de antrenament de dimensiuni diferite. Documentele pot avea număr diferit de atribute, adică două exemple din același set de date pot avea dimensiuni diferite, de exemplu un document poate avea 1300 de atribute, iar altul 1000. Valoarea lui **m**, unde **m** este numărul de coloane din matrice, este numărul maxim de atribute pe care îl poate avea un exemplu. Dacă un atribut nu apare într-un exemplu o să se scrie „0” în matrice, deoarece atributul nu are valoare în acel exemplu, astfel exemplele o să aibă aceeași lungime atunci când o să se aplice metrica de similaritate. Toate exemplele o să fie de lungime **m** înainte de a se aplica metrica de similaritate.

Această structură a fost aleasă și din dorința de optimizare a memoriei pe care datele de intrare o ocupă. Din acest motiv s-a ales tipul de date Int16. Încă un motiv a fost și complexitatea redusă de acces a elementelor din structură, astfel a rezultat și o mică optimizare de timp.

Datorită acestei etape, procesul poate folosi două tipuri diferite de fișiere. Tipul fișierului este dat de conținutul pe care acesta îl are, adică este dat de datele pe care le conține. Datele din fișiere sunt exemplele cu care lucrează algoritmul, iar acestea sunt de două tipuri: exemple care reprezintă puncte pe grafic și exemple care reprezintă documente sub formă de vectori. Este important de specificat că procesul nu poate folosi ambele fișiere simultan. Procesul folosește un tip de fișier până la finalul acestuia, dacă se dorește alt tip, trebuie să se refacă tot procesul. Acesta nu poate fi aplicat în același timp și pe puncte și pe documente, trebuie să se aplice ori pe puncte ori pe documente.

Tipurile și conținutul fișierelor cu care lucrează procesul sunt următoarele:

- **Fișiere care conțin puncte pe grafic:** aceste fișiere conțin exemple care reprezintă puncte pe grafic, deci dimensiunea exemplelor o să fie doi. Exemplele au dimensiunea doi, prin

urmare numărul de atribute este doi. Atributele exemplelor din acest tip de fișier sunt coordonata pe axa X și coordonata pe axa Y de pe grafic. Aceste fișiere arată în felul următor:

```
X,Y,GRUP
178,169,0
-157,-108,3
-142,-101,3
-146,-106,3
-34,34,2
25,-101,1
-59,21,2
-153,-110,3
-54,26,2
...
```

Fig. 3.2 Exemplu de fișier care conține puncte pe grafic

Fiecare linie din acest tip fișier reprezintă un exemplu care conține trei informații. X este valoarea punctului pe axa X a graficului, iar Y este valoarea punctului pe axa Y a graficului. Aceste două informații sunt atributele exemplului. GRUP reprezintă eticheta inițială a exemplului, iar cu ajutorul acesteia se calculează metricile externe.

- **Fișiere care conțin documente sub formă de vectori:** conținutul fișierelor a rezultat după etapa de preprocesare a bazei de date de la ziarul Reuters. Etapa de preprocesare se mai numește și etapa de extragere a trăsăturilor. Această etapă, pe baza conținutului documentelor, extrage anumite trăsături și atribute pe care le are fiecare document. Aceste fișiere conțin documente reprezentate vectorial prin vectori care conțin frecvența de apariție a cuvintelor, deci fiecare vector reprezintă un document. Conținutul acestor fișiere arată în felul următor:

#Samples 4702

#Attributes 1309

#Topics 24

@attribute 0 1.0

@attribute 1 2.0

@attribute 2 3.0

@attribute 4 4.0

...

@attribute 9181 1309.0

@topic c18 747

...

@topic c1511 410

@data

#document1

0:1 1:8 5:1 7:5 9:1 10:1 12:1 15:2 29:3 34:19 39:1 41:1 55:1 57:5 60:2 61:11 62:1 67:3 68:1 73:3 82:1 91:1 108:2 114:1 115:3 119:1 128:1 135:2 146:1 152:1 158:1 168:1 173:1 176:1 180:1 189:1 190:2 194:1 202:2 218:1 223:1 224:1 228:2 230:1 277:1 280:1 285:1 301:1 307:1 318:2 325:1 330:4 332:1 341:1 342:1 343:1 353:1 366:1 367:1 380:1 439:1 472:1 473:2 478:1 483:1 487:1 493:2 541:1 556:2 560:7 574:1 586:1 588:1 646:3 656:2 694:2 723:1 747:1 760:1 762:1 873:1 901:1 906:1 912:1 939:1 949:2 950:1 967:1 978:1 1014:3 1047:1 1092:1 1169:1 1243:1 1295:2 # c31

#document2

0:1 1:1 7:5 15:1 17:2 22:1 38:1 40:1 42:1 45:9 47:1 55:7 60:1 66:1 68:1 70:1 74:1 77:1 89:1 91:1 111:2 121:1 135:1 149:1 154:1 159:1 174:1 183:1 200:1 211:1 215:1 228:1 237:1 246:1 251:2 260:1 277:2 278:1 331:4 341:1 345:4 349:2 362:2 365:5 371:2 375:1 378:1 384:1 385:2 419:2 446:2 458:1 463:1 465:1 475:1 478:1 515:3 528:3 544:1 571:1 573:1 593:1 599:4 622:4 624:1 636:1 672:1 680:1 687:1 704:1 718:1 719:2 722:1 736:1 743:1 747:1 764:1 782:1 793:1 810:1 864:1 908:1 925:2 956:1 1005:1 1014:1 1150:1 1163:2 # c14 c17 c171

...

Fig. 3.3 Exemplu de fișier care conține documente sub formă de vectori

Aceste fișiere conțin mai multe informații decât fișierele care conțin puncte:

- **#Samples**...: reprezintă numărul de documente, adică numărul de linii din matricea de reprezentare a datelor.
- **#Attributes**...: reprezintă numărul maxim de atribute pe care în poate avea un exemplu. Acesta este și numărul de coloane din matricea de reprezentare a datelor.
- **#Topics**...: reprezintă numărul de etichete din acest fișier;
- **@data**: marchează începutul documentelor;
- Vectorii sunt formați din perechi de forma „atribut : frecvență”, în care „atribut” este indexul unui cuvânt dintr-un dicționar global din etapa de extragere a trăsăturilor și „frecvență” repezintă de câte ori a apărut acel cuvânt în documentul respectiv (exemplu 7:5, atributul 7 apare de 5 ori în document);
- La finalul fiecărui vector din acest fișier avem un „#” care marchează sfârșitul vectorului, iar după „#” avem eticheta originală a documentului (exemplu #c31).

În această etapă fișierele trebuie sa fie citite linie cu linie și trebuie să se extraga doar anumite valori din aceste fișiere. Etapa conține două implementări de citire din fișier, una pentru fișierele care conțin puncte și una pentru fișierele care conțin documente sub formă de vectori.

Pentru a putea face o analiză ulterioară a algoritmului de clustering folosit în acest proces, trebuie să salvăm grupurile sau etichetele inițiale din documente. Grupurile sau etichetele inițiale se salvează într-un vector în care indexul fiecărui element din acesta este identic cu indexul fiecărei linii din matrice. În acest fel memorăm fiecare document din ce grup a făcut parte inițial sau ce etichetă a avut.

Fișierele care conțin puncte sunt fișiere care conțin exemple cu două dimensiuni. Implementarea constă în citirea tuturor liniilor din fișier, cu excepția primei linii. La citirea fiecărei linie din fișier trebuie să se și se despart elementele din linie după virgulă pentru a extrage cele trei valori care caracterizează exemplul, adică cele două atribute și eticheta inițială a acestuia. Valorile sunt salvate într-o listă de forma: *List<(string x, string y, string z)> lines = new List<(string, string, string)>(),* unde x reprezintă valoare X citită din fișier, y reprezintă valoarea Y din fișier și z reprezintă valoarea GRUP din fișier. Despărțirea elementelor se face prin aplicarea metodei *Split(',')* pe care o are tipul string. Această metodă returnează un vector de tipul string (*string[] temp*) cu cele trei elemente. Fiecare element din linie, o să fie pus în `lines` în zona în care trebuie (*lines.Add((temp[0], temp[1], temp[2]))*). După ce am creat această listă am scris valorile elementelor x, y în matricea prin care se reprezintă datele și valorile elementului z, etichetele inițiale, într-un vector de tip string. Important este ca indexul liniei din matrice să fie același cu

indexul vectorului de etichete, pentru a se cunoaște care a fost eticheta inițială a fiecărui exemplu. Un exemplu de implementare in limbajul C#:

```
1 reference
public void Read2DFile(out Int16[,] values, out String[] initialCluster, string filePath)
{
    string line;
    List<(string x, string y, string z)> lines = new List<(string, string, string)>();
    StreamReader file = new StreamReader(filePath);

    line = file.ReadLine();//citesc prima linie si dupa o sterg
    line = "";

    int numberOfLines = 0;
    while ((line = file.ReadLine()) != null)
    {
        string[] temp = line.Split(',');
        lines.Add((temp[0], temp[1], temp[2]));
        numberOfLines++;
    }

    values = new Int16[numberOfLines, 2];
    initialCluster = new String[numberOfLines];

    for (var i = 0; i < numberOfLines; i++)
    {
        for (var j = 0; j < 3; j++)
        {
            if (j == 0)
            {
                values[i, j] = Convert.ToInt16(lines[i].x);
            }
            else if (j == 1)
            {
                values[i, j] = Convert.ToInt16(lines[i].y);
            }
            else
            {
                initialCluster[i] = lines[i].z;
            }
        }
    }
}
```

Code 3.1 Metodă de citire pentru fișierele care conțin puncte

Această metodă realizează citirea fișierelor care conțin puncte. Parametrii pe care îi primește această metodă sunt:

- ***out*** *Int16[,] values*: este matricea prin care sunt reprezentate datele;
- ***out*** *String[] initialCluster*: este vectorul în care se salvează etichetele inițiale;
- *string filePath*: este calea către fișierul care trebuie citit.

Parametrii care au „***out***” în față sunt parametrii de ieșire, adică aceștia trebuie inițializați în această metodă și modificările făcute acestora nu sunt șterse la finalul metodei.

Fișierele care conțin documente sub formă de vectori sunt fișiere conțin exemple cu N dimensiuni. Se începe prin citirea primelor două linii care sunt numărul de exemple și numărul maxim de atribute pe care îl poate avea un exemplu.

```
#region Citeste numaurl de lini si de coloane din fisier

string line1 = file.ReadLine();
string line2 = file.ReadLine();

string[] vectorLine1 = line1.Split(' ');
string[] vectorLine2 = line2.Split(' ');

int numberOfLines = Convert.ToInt32(vectorLine1[1]);
int numberOfColumns = Convert.ToInt32(vectorLine2[1]);
#endregion
```

Code 3.2 Citire numărul de documente și numărul de atribute

După ce se citesc numărul de linii și de coloane, se declară o listă de tipul string în care sunt salvate linii din fișier (*List<string> lines = new List<string>()*). În această listă sunt salvate liniile care nu încep cu anumite caractere sau liniile care nu sunt goale. Elementele din listă o să conțină vectorii de frecvență și prima etichetă a fiecărui exemplu.

```
1 reference
public void ReadNDFile(out Int16[,] matrix, out String[] initialCluster, string filePath)
{
    string line;
    List<string> lines = new List<string>(); //fiecare linie din string
    StreamReader file = new StreamReader(filePath);

    Citeste numaurl de lini si de coloane din fisier

    while ((line = file.ReadLine()) != null)
    {
        if ((line.StartsWith("@")) || (line.StartsWith("#")) || (line == ""))
        {
            //Nu fa nimic, nu imi introdu in lista de string
        }
        else
        {
            int indexSpecificChar = line.IndexOf("#");
            string tempLine = line.Substring(0, indexSpecificChar);

            string clust = line.Substring(indexSpecificChar + 2, ((line.Length - tempLine.Length) - 2));
            int indexCharForCluster = clust.IndexOf(' ');
            string tempClust = clust.Substring(0, indexCharForCluster);

            tempLine = tempLine + " " + tempClust;
            lines.Add(tempLine);
        }
    }
    Tuple<Int16[,], String[]> temp= CreateDataMatrixAndClustersVector(lines, numberOfLines, numberOfColumns);
    matrix = temp.Item1;
    initialCluster = temp.Item2;
}
```

Code 3.3 Metodă care citește fișierele care conțin documente sub formă de vectori

Această metodă primește ca parametrii *matrix* care este matricea prin care sunt reprezentate datele, *initialCluste* care este vectorul în care se salvează etichetele inițiale și *filePath* este calea către fișierul care trebuie citit.

La finalul citiri liniilor, care sunt vectori de frecvență, din fișier se apelează metoda *CreateDataMatrixAndClustersVector(lines, numberOfLines, numberOfColumns)* care returnează un *Tuple<Int16[,], String[]>*, în care Item1 este matricea prin care sunt reprezentate datele și Item2 este vectorul de grupuri sau etichete inițiale. Această metodă primește ca parametri lista de linii citite din fișier și numărul de linii și coloane pe care trebuie să le aibă matricea de la Item1 din tuplul returnat. Metoda prelucrează fiecare element din lista de lini și scrie în matrice la indexul liniei frecvența de apariție a atributelor. Atributele sunt pe coloana, iar dacă un atribut nu apare în linia citită, această metodă scrie 0 în matrice la coloana atributului care nu apare. A finalul liniei se găsește eticheta, acesta este salvată într-un vector de tipul string.

```
//Functia aceasta creaza matricea de marime NxN cu valorile citite din fisier si vectorul de clustere din fisier
1 reference
public Tuple<Int16[,], String[]> CreateDataMatrixAndClustersVector(List<string> lines, int numberOfLines, int numberOfColumns)
{
    Int16[,] matrix = new Int16[numberOfLines, numberOfColumns];
    String[] vector = new String[numberOfLines];

    for (var indexLine = 0; indexLine < lines.Count; indexLine++)
    {
        string[] lineSplit = lines[indexLine].Split(' ');
        vector[indexLine] = lineSplit[lineSplit.Length - 1];
        for (var indexLineSplit = 0; indexLineSplit < lineSplit.Length - 2; indexLineSplit++)
        {
            string[] valueSplit = lineSplit[indexLineSplit].Split(':');
            int key = Convert.ToInt32(valueSplit[0]);
            Int16 value = Convert.ToInt16(valueSplit[1]);

            matrix[indexLine, key] = value;
        }
    }

    return Tuple.Create(matrix,vector);
}
```

Code 3.4 Metodă care creează matricea și vectorul cu etichete

3.3 Etapa de normalizare a datelor

Pentru ca face algoritmii de clustering să fie cât mai eficienți se recomandă aplicarea anumitor algoritmi de normalizare, deoarece dorim ca datele cu care lucrează algoritmul de clustering să fie cât mai convenabile pentru acesta.

Se recomandă aplicare normalizării, deoarece algoritmii de clustering grupează exemplele pe baza de similaritatea sau de disimilaritatea, iar acestea sunt calculate cu ajutorul distanțelor. Datorită normalizării, distanțele dintre obiectele pe care algoritmul dorește să le grupeze sunt într-un domeniu controlabil. Prin îngustarea domeniului, distanțele dintre obiectele pe care dorim să le grupăm sunt mai precise, adică avem o similaritate sau disimilaritate mai concretă între obiectele pe care dorim să le grupăm.

Scopul principal al acestei etape este acela de a schimba domeniul original de reprezentare a datelor într-un domeniu nou pe care îl cunoaștem. Această etapă creează o matrice de date normalizate pe baza matricei rezultate după etapa de citire a datelor. Aplicarea acestei etape este foarte importantă în situațiile în care datele au același înțeles dar nu sunt distribuite la fel. În cazul documentelor frecvența de apariție a cuvintelor influențează mult similaritate, dar dacă cuvintele sunt la fel în două documente, dar în unul cuvintele apar de mai multe ori din cauză că sunt copiate de mai multe ori. În această situație înțelesul documentelor este același, dar distanța dintre documente este mare, iar algoritmul o să le grupeze diferit, deoarece acesta consideră că nu sunt similare.

Există multe tipuri de normalizare a datelor. Tipul pe care îl folosim atunci când dorim să facem normalizarea datelor trebuie să fie aleasă în funcție de cum sunt reprezentate datele inițial. Aceasta trebuie sa fie aleasă și în funcție de ce reprezintă datele pe care dorim să le normalizăm. Trebuie să se țină cont și de felul în care dorim să restrângem domeniul de reprezentare a datelor sau care vrem să fie noul domeniu în care sunt reprezentate datele, respectiv noul interval în care sunt regăsite datele.

Această etapă implementează mai multe tipuri de normalizare, deoarece dorim să facem o analiză a celor doi algoritmi de clustering fiind aplicați pe date normalizate cu mai multe tipuri. Dorim să vedem ce tip de normalizare este mai bună pentru fiecare algoritm și să facem o comparație a algoritmilor care folosesc același tip, deoarece dorim să vedem pentru ce algoritm tipul a fost mai eficientă.

Anumite tipuri de normalizare nu pot fi aplicate pe date negative. Datorită faptului că unele coordonate ale punctelor de pe grafic pot avea valori negative și anumite tipuri nu pot fi aplicate pe date negative, am făcut o „ajustare” a intervalului, o rescalare a intervalului într-un domeniu doar cu valori pozitive. Această ajustare se face doar atunci când se normalizeaza date care reprezintă puncte de pe grafic. Am făcut o analiză asupra domeniului din care fac parte punctele și am aflat că punctele pot avea valori în intervalul [-200,200]. Datorită acestei analize am realizat că dacă adunăm 200 (*inputData[i, j]+ajustare*) la fiecare valoare putem face trecerea într-un interval pozitiv, în intervalul [200,400]. Această trecere se face cu ajutorul unei variabile „*int ajustare*”. Valoarea variabilei este 200 când încărcăm date care reprezintă puncte pe grafic și 0 când încărcăm date care reprezintă frecvența de apariție a cuvintelor în documente. Variabila este trimisă ca parametru metodelor care implementează tipuri de normalizare care nu pot să fie aplicate pe date care conțin valori negative (*NormalizareSumaUnu(Int16[,] inputData, int ajustare)*).

În cele ce urmează am să arăt ce tipuri de normalizare am folosite în această etapă și cum au fost acestea implementate.

- **Normalizarea binară**

```
1 reference
public double[,] NormalizareBinara(Int16[,] inputData)
{
    double[,] returnMatrix = new double[inputData.GetLength(0), inputData.GetLength(1)];

    for (var i = 0; i < inputData.GetLength(0); i++)
    {
        for (var j = 0; j < inputData.GetLength(1); j++)
        {
            if (inputData[i, j] == 0)
            {
                returnMatrix[i, j] = 0;
            }
            else
            {
                returnMatrix[i, j] = 1;
            }
        }

    }

    return returnMatrix;
}
```

Code 3.5 Metodă care realizează normalizare binară

Această metodă parcurge matricea primită ca parametru și verifică dacă valoarea atributului este 0 și nu o schimbă, dar dacă este diferită atunci o să fie schimbată în 1. Această metodă nu ține cont de valoarea atributului, verifică doar dacă apare sau nu în exemplu. Unde apara valoarea 1 atunci atributul apare în exemplu, dacă apare 0 atunci atributul nu apare.

- **Normalizarea Cornell-Smart**

```
1 reference
public double[,] NormalizareCornel_Smart(Int16[,] inputData,int ajustare)
{
    double[,] returnMatrix = new double[inputData.GetLength(0), inputData.GetLength(1)];

    for (var i = 0; i < inputData.GetLength(0); i++)
    {
        for (var j = 0; j < inputData.GetLength(1); j++)
        {
            if (inputData[i, j] == 0)
            {
                returnMatrix[i, j] = 0;
            }
            else
            {
                returnMatrix[i, j] = (double)1 + Math.Log((double)1 + Math.Log((inputData[i, j]+ajustare), 10), 10);
            }

        }
    }

    double test1 = GetMaxValueFromMatrix(returnMatrix);
    double test2 = GetMinValueFromMatrix(returnMatrix);

    MessageBox.Show("Maximul dupa normalizare este: " + test1.ToString() +
                    "\nMinimul dupa normalizare este: " + test2.ToString());
    return returnMatrix;
}
```

Code 3.6 Metodă care realizează normalizare Cornell-Smart

Parametrii pe care îi primește această metodă sunt matricea de date rezultată după etapa de citire a datelor și variabila *ajustare*. Se parcurge matrice primită ca parametru și se verifică dacă elementul are valoarea 0, dacă valoarea este diferită de 0, atunci se aplică o formulă asupra elementului respectiv (*returnMatrix[i, j] = (double)1 + Math.Log((double)1 + Math.Log((inputData[i, j]+ajustare), 10), 10)*). Această metodă returnează o matrice de date normalizate.

- **Normalizarea Suma1**

```
1 reference
public double[,] NormalizareSumaUnu(Int16[,] inputData, int ajustare)
{
    double[,] returnMatrix = new double[inputData.GetLength(0), inputData.GetLength(1)];
    int sum = 0;
    List<int> vectorSuma = new List<int>();
    for (var i = 0; i < inputData.GetLength(0); i++)
    {
        for (var j = 0; j < inputData.GetLength(1); j++)
        {

            sum += (inputData[i, j]+ajustare);
        }
        vectorSuma.Add(sum);
        sum = 0;
    }

    for (var i = 0; i < inputData.GetLength(0); i++)
    {
        for (var j = 0; j < inputData.GetLength(1); j++)
        {
            returnMatrix[i, j] = (double)(inputData[i, j]+ajustare) / (double)vectorSuma[i];
        }
    }
    double test1 = GetMaxValueFromMatrix(returnMatrix);
    double test2 = GetMinValueFromMatrix(returnMatrix);

    MessageBox.Show("Maximul dupa normalizare este: " + test1.ToString() +
                    "\nMinimul dupa normalizare este: " + test2.ToString());
    return returnMatrix;
}
```

Code 3.7 Metodă care realizează normalizare suma1

Această metodă face de două ori parcurgerea datelor, odată pentru a face suma valorilor atributelor fiecărui exemplu și o dată pentru a împărți fiecare atribut din exemplu la suma calculată pe acel exemplu. Exemplele sunt linile din matricea rezultată după etapa de citire a datelor. Suma pe fiecare exemplu este salvată într-o listă *vectorSum*. După ce se creează această listă se împarte fiecare atribut al exemplului la suma calculată pe acel exemplu (*returnMatrix[i, j] = (double)(inputData[i, j]+ajustare) / (double)vectorSuma[i]*). Indexul „i” al matricei *inputData* este același cu indexul „i” din lista *vectorSum*, astfel știm care este suma fiecărui exemplu. Metoda primește ca parametrii matricea rezultată la finalul etapei de citire a datelor (*inputData*) și variabila *ajustare*, iar aceasta returnează o matrice care conține datele normalizate. La acest tip de

normalizare nu se cunosc limitele intervalului rezultat după aplicare, deoarece la sfârșitul normalizări suma pe exemplu trebuie să fie 1, adică limita intervalului poate să fie 1 dacă exemplul are un singur atribut.

- **Normalizarea nominală**

Acest tip de normalizare este asemănător cu tipul prezentat anterior. După aplicarea acestui tip de normalizare nu se cunosc limitele intervalului rezultat după aplicare. Acest tip trebuie să parcurgă datele de două ori, odată pentru a determina maximul de pe fiecare exemplu și o dată pentru a împarți fiecare atribut al exemplului la maximul acestuia.

```
1 reference
public double[,] NormalizareNominala(Int16[,] inputData, int ajustare)
{
    double[,] returnMatrix = new double[inputData.GetLength(0), inputData.GetLength(1)];
    List<int> dataDocument = new List<int>();
    List<int> listMaxDocument = new List<int>();
    for (var i = 0; i < inputData.GetLength(0); i++)
    {
        for (var j = 0; j < inputData.GetLength(1); j++)
        {
            dataDocument.Add((inputData[i, j]+ajustare));
        }
        listMaxDocument.Add(GetMaxValueFromList(dataDocument));
        dataDocument.Clear();
    }
    for (var i = 0; i < inputData.GetLength(0); i++)
    {
        for (var j = 0; j < inputData.GetLength(1); j++)
        {
            returnMatrix[i, j] = (double)(inputData[i, j]+ajustare) / (double)listMaxDocument[i];
        }
    }
    double test1 = GetMaxValueFromMatrix(returnMatrix);
    double test2 = GetMinValueFromMatrix(returnMatrix);

    MessageBox.Show("Maximul dupa normalizare este: " + test1.ToString() +
                    "\nMinimul dupa normalizare este: " + test2.ToString());
    return returnMatrix;
}
```

Code 3.8 Metodă care realizează normalizare nominală

Parametrii de intrare pe care îi primește această metodă sunt: matricea rezultată după etapa de citire a datelor (*inputData*) și variabila *ajustare*.Această metodă parcurge matricea primită ca parametru de două ori. La prima pacurgere a matricei se determină maximul de pe fiecare linie a matricei, adică atributul cu valoarea maximă din fiecare exemplu. Fiecare maxim este salvat într-o listă care are indexul identic cu linia din matrice, pentru a se cunoaște din ce exemplu face parte acel maxim. După ce au fost determinate maximele de pe toate liniile se mai parcurge odată matricea pentru a împărți fiecare atribut la acel maxim (returnMatrix[i, j] = (double)(inputData[i, j]+ajustare) / (double)listMaxDocument[i]). Această metodă returnează o matrice de date normalizate.

- **Normalizarea Min-Max**

Metodă care realizează normalizare min-max primește ca parametrii matricea pe care trebuie să se aplice normalizarea și noile limite ale intervalului din care vor face parte datele normalizate.

```
1 reference
public double[,] NormalizareMinMax(Int16[,] inputData, Int16 newMin, Int16 newMax)
{
    double[,] returnMatrix = new double[inputData.GetLength(0), inputData.GetLength(1)];

    Int16 maxData = GetMaxValueFromMatrix(inputData);
    Int16 minData = GetMinValueFromMatrix(inputData);

    for (var i = 0; i < inputData.GetLength(0); i++)
    {
        for (var j = 0; j < inputData.GetLength(1); j++)
        {
            returnMatrix[i, j] = (((double)(inputData[i, j] - minData)) /
                                  ((double)maxData - minData)) * ((double)(newMax - newMin)) + newMin;
        }
    }

    double test1 = GetMaxValueFromMatrix(returnMatrix);
    double test2 = GetMinValueFromMatrix(returnMatrix);

    MessageBox.Show("Maximul dupa normalizare este: " + test1.ToString() +
                    "\nMinimul dupa normalizare este: " + test2.ToString());
    return returnMatrix;
}
```

Code 3.9 Metodă care realizează normalizare min-max

Primul pas pe care îl face această metodă este de a determina minimul și maximul actual din matrice, adică limitele intervalului din care fac parte datele originale. După determinarea limitelor se aplică o formulă pe fiecare valoare din această matrice (returnMatrix[i, j] = (((double)(inputData[i, j] - minData)) / ((double)maxData - minData)) * ((double)(newMax - newMin)) + newMin). Această metodă returnează o matrice de date normalizate.

Toate matricele rezultate după aplicarea metodelor care realizează normalizarea datelor au aceeași dimensiune ca matricea rezultată după etapa de citire a datelor.

3.4 Etapa de calcul a matricei de similaritate

În ceea ce urmează doresc să descriu etapa de calcul a matriceri de similaritate dintre exemple. O să descriu de ce este folositoare această etapă, ce reprezintă această etapă și cum a fost implementată aceasta.

Algoritmii de clustering creează grupe formate din exemple. Formarea grupelor se face pe baza asemănărilor dintre exemple, adică dacă două exemple sunt asemănătoare aceste vor face parte din același grup. Pentru a determina dacă două exemple sunt asemănătoare sau nu, adică dacă

fac sau nu parte din același cluster, trebuie să determinăm similaritatea dintre exemple. Pentru a determina similaritatea dintre exemple trebuie să calculăm distanța dintre acestea.

Această etapă creează, pe baza matricei rezultate după etapa de normalizare, o matrice de similaritate. Această matrice de similaritate conține similaritățile dintre toate exemplele care trebuie grupate. Matricea de similaritate conține distanța între fiecare exemplu cu toate celelalte exemple. Această matrice conține pe linia „i" distanțele între exemplul „i" și toate celelealte exemple. Aceasta este o matrice de n×n, în care n reprezintă numărul de exemple pe cae algoritmul trebuie să le grupeze.

Scopul acestei etape este acela de a crea o matrice care să conțină toate distanțele și pe care algoritmul de clustering să o folosească. Datorită acestei etape se scade și computația, deoarece toate distanțele sunt calculate într-o singură etapă, nu să se apeleze metodele care realizează calculul distanței de fiecare dată când algoritmul are nevoie de ea. Matricea de similaritate este necesară când se folosește algoritmul HAC.

Această etapă implementează mai multe tipuri de calcul a distanțelor, deoarece algoritmii de clustering sunt foarte influențați de cum sunt calculate distanțele. Sunt implementate mai multe pentru a analiza ce distanță este mai bună pentru un algoritm și să facem o comparație între cei doi algoritmi când folosesc acea distanță. Se face și o analiză pentru a vedem de distanță este mai bună în funcție de datele pe care se calculează și ce distanță este bună pentru cei doi algoritmi.

În ceea ce urmează am să prezint tipurile de distanțe folosite în acest proces și cum au fost implementate acestea.

Datorită faptului că Distanța Minkowski este o generalizare a Distanței Euclidiene și a Distanței Manhattan, am folosit o singură metodă care implementează toate cele trei distanțe.

```
1 reference
private double DistantaMinkowski(int order, double[] firstPoint, double[] secondPoint)
{
    double sum = 0;
    int dimension = firstPoint.Length;
    for (var i = 0; i < dimension; i++)
    {

        sum += Math.Pow(Math.Abs(firstPoint[i] - secondPoint[i]), order);
    }

    return Math.Pow(sum, (double)1 / order);
}
```

Code 3.10 Metoda care calculează cele trei distanțe

Această metodă primește ca parametrii doi vectori de dimensiuni egale și un parametru *int order*. Metoda calculează distanța între cei doi vectori și o returnează. Prin parametru order

decidem care din cele trei distanțe este calculată. Dacă order are valoarea „1” atunci o să se calculeze distanța Manhattan, dacă are valoarea „2” o să se calculeze distanța Euclidiană, iar dacă *order* are o valoare mai mare se calculează distanța Minkowski de ordinul respectiv.

Distanța cosinus calculează disimilaritatea dintre două exemple și are valori în intervalul [0,1]. Datorită faptului că această metodă calculează disimilaritatea și trebuie să creăm matricea de similaritate, se face un artificiu. Se scade din 1 distanța cosinus, astfel din disimilaritate facem o conversie la similaritate.

```
1 reference
private double DistantaCosinus(double[] firstPoint, double[] secondPoint)
{
    double sum1 = 0;
    double sum2 = 0;
    double sum3 = 0;
    int dimension = firstPoint.Length;
    for (var i = 0; i < dimension; i++)
    {
        sum1 += firstPoint[i] * secondPoint[i];
        sum2 += Math.Pow(firstPoint[i], 2);
        sum3 += Math.Pow(secondPoint[i], 2);
    }
    sum2 = Math.Sqrt(sum2);
    sum3 = Math.Sqrt(sum3);
    sum1 = sum1 / (sum2 * sum3);

    if ((sum1 < -0.001) || (sum1 > 1.1))
    {
        MessageBox.Show("Error");
    }

    return 1 - sum1;
}
```

Code 3.11 Metoda care calculează distanța cosinus

Metoda primește ca și parametrii doi vectori și returnează 1 minus distanța cosinus dintre aceștia. Sintaxa cu „if” care se află înainte de *return* este folosită pentru a verifica dacă valoarea se află sau nu în spațiul [0,1].

În această metodă se calculează suma produsului dintre elementele celor doi vectori și suma elementelor ridicate la puterea a doua din cei doi vectori. La final prima sumă calculată este împărțită la produsul dintre sumele elementelor ridicate la puterea a doua.

În ceea ce urmează am să prezint si metoda care creează matricea de similaritate cu ajutorul metodelor prezentate anterior.

```
1 reference
public double[,] GetDistanceMatrix(ref double[,] normalizedData, int tipDistanta)
{
    Stopwatch sw = new Stopwatch();

    double[,] distanceMatrix = new double[normalizedData.GetLength(0), normalizedData.GetLength(0)];
    sw.Start();

    double[] firstPoint = new double[normalizedData.GetLength(1)];
    double[] secondPoint = new double[normalizedData.GetLength(1)];
    int nrCol = normalizedData.GetLength(1);
    double tempDist = 0;
    if (tipDistanta == -1)
    {
        for (int i = 0; i < distanceMatrix.GetLength(0); i++)
        {
            System.Buffer.BlockCopy(normalizedData, sizeof(double) * nrCol * i, firstPoint, 0, sizeof(double) * nrCol);
            for (int j = i + 1; j < distanceMatrix.GetLength(0); j++)
            {
                System.Buffer.BlockCopy(normalizedData, sizeof(double) * nrCol * j, secondPoint, 0, sizeof(double) * nrCol);
                tempDist = DistantaCosinus(firstPoint, secondPoint);
                distanceMatrix[i, j] = tempDist;
                distanceMatrix[j, i] = tempDist;
                if (distMax < distanceMatrix[i, j])
                {
                    distMax = distanceMatrix[i, j];
                }
            }
        }
    }
    else
    {
        for (int i = 0; i < normalizedData.GetLength(0); i++)
        {

            System.Buffer.BlockCopy(normalizedData, sizeof(double) * nrCol * i, firstPoint, 0, sizeof(double) * nrCol);
            for (int j = i + 1; j < normalizedData.GetLength(0); j++)
            {

                System.Buffer.BlockCopy(normalizedData, sizeof(double) * nrCol * j, secondPoint, 0, sizeof(double) * nrCol);


                tempDist = DistantaMinkowski(tipDistanta, firstPoint, secondPoint);
                distanceMatrix[i, j] = tempDist;
                distanceMatrix[j, i] = tempDist;
                if (distMax < distanceMatrix[i, j])
                {
                    distMax = distanceMatrix[i, j];
                }
            }
        }
    }
    sw.Stop();

    Console.WriteLine("Matrice Distanta Run Time: {0}", sw.Elapsed);

    return distanceMatrix;
}
```

Code 3.12 Metoda care creează matricea de similaritate

Parametri pe care îi primește această metodă sunt matricea de date normalizate și o variabilă care indică ce tip de distanță este folosit. Sintxal *ref* din fața variabilei care reprezintă matricea, ne arată că nu o să se facă o copie a matricei când este apelată această metodă, calculele for fi făcute direct pe matricea originala primită ca referință. Folosind *ref* am reușit să fac o reducere de memorie, deoarece când se apela metoda se făcea o copie a matricei care era ștearsă la ieșirea din metodă. Memoria pe care o consuma matricea se dubla în timpul execuției acestei

metode. Această metodă returnează matricea de similaritate pe care a fost calculată cu ajutorul parametrilor primiți.

3.5 Etapa de clustering

Etapa de clustering este etapa în care este aplicat unul din cei doi algoritmi asupra exemplelor pe care dorim să le grupăm. În această etapă se introduc și parametrii cu care lucrează algoritmul de clustering, astfel ne oferă o libertate și o flexibilitate în aplicarea algoritmilor și analiza acestora. În ceea ce urmează am să prezint algoritmii care fac parte din această etapă și cum au fost aceștia implementați.

3.5.1 DBSCAN

Acest algoritm grupează datele pe baza de densitate. Pentru a implementa acest algoritm am creat o clasă *DBSCAN* care conține metode prin care se implementează acest algoritm.

```
2 references
class DBSCAN
{

    private int distanceType;
    private double[,] distanceMatrix;

    0 references
    public int DistanceType { get => distanceType; set => distanceType = value; }

    1 reference
    public DBSCAN(ref double[,] distanceMatrix)
    {
        this.distanceMatrix = distanceMatrix;
    }

    1 reference
    public void Implementation(out List<Data> dataList,  double eps, int minPts)...
    1 reference
    private void GetClusters(List<Data> dataList, double eps, int minPts)...
    1 reference
    private bool ExpandCluster(List<Data> dataList, Data p, int clusterId, double eps, int minPts)...
    2 references
    private List<Data> GetRegion(List<Data> dataList, Data p, double eps)...
}
```

Code 3.13 Clasa DBSCAN

În această clasă se face o referință la matricea de distanțe calculată în etapa de calcul a matricei de similaritate. Cu ajutorul acestei matrici se verifică distanța între exemple și cu ajutorul acesteia se creează densitatea. Această clasă implementează patru metode cu ajutorul cărora se aplică algoritmul DBSCAN și se returnează o listă care conține exemplele și din ce cluster face parte fiecare exemplu.

```
1 reference
public void Implementation(out List<Data> dataList,  double eps, int minPts)
{
    //lista care contine id/val liniei de la fiecare linie din mat/document
    dataList = new List<Data>();

    //aceasta linie intoarce numarul de linii din matrice
    int numberOfDocumnet = distanceMatrix.GetLength(0);

    //creaza o lista cu toate dic dn matrice si il marcheaza ca fiind neclasterizat
    for (var i = 0; i < numberOfDocumnet; i++)
    {
        dataList.Add(new Data(i));
    }

    GetClusters(dataList, eps, minPts);

}
```

Code 3.14 Metoda care implementează algoritmul DBSCAN

Această metodă primește ca parametrii o listă care la finalul algoritmului o să conțină exemplele și clusterul din care face parte fiecare exemplu, distanța minimă între două puncte (*eps*) și numărul minim de puncte care pot forma o densitate sau un cluster (*minPts*). Această metodă începe prin inițializarea listei primite ca parametru, deoarece aceasta este un parametru de ieșire, datorită sintaxei *out* din fața acestuia. După inițializarea listei trebuie să se introducă toate exemplele în aceasta, exemplele fiind etichetate ca neclusterizate. După ce a fost populată lista se apelează altă metodă prin intermediul căreia se clusterizează datele.

```
1 reference
private void GetClusters(List<Data> dataList, double eps, int minPts)
{
    Random rand = new Random();
    List<int> indexList = new List<int>();

    int clusterId = 1;//Initializez prima valuare a clusterului, adica primul cluster

    //Aici se amesteca lista de index de mai sus
    indexList = Enumerable.Range(0, dataList.Count).OrderBy(c => rand.Next()).ToList();

    foreach (var i in indexList)
    {
        Data p = dataList[i];
        if (p.ClusterId == DataProperties.UNCLUSTERED)
        {
            if (ExpandCluster(dataList, p, clusterId, eps, minPts)) clusterId++;
        }
    }
}
```

Code 3.15 Metoda care grupează datele în algoritmul DBSCAN

La intrarea în această metodă se creează o listă de index cu ajutorul căreia amestecăm random elementele din listă și putem începe algoritmul cu un exemplu aleatoriu. După se inițializează valoarea primului cluster cu 1 și se începe parcurgerea exemplelor aleator. Se ia un exemplu aleator și se verifică dacă este sau nu clusterizat, dacă nu este se apelează metoda care extinde clusterul, metoda care începe să creeze densitatea. După ce a fost aplicată această metodă id-ul clusterului este incrementat, deoarece la finalul acestei metode înseamnă ca nu s-a mai putut extinde densitatea.

```
1 reference
private bool ExpandCluster(List<Data> dataList, Data p, int clusterId, double eps, int minPts)
{
    List<Data> seeds = GetRegion(dataList, p, eps);
    if (seeds.Count < minPts) // no core point
    {
        foreach (var i in seeds)
        {
            i.ClusterId = DataProperties.NOISE;
        }
        return false;
    }
    else // all points in seeds are density reachable from point 'p'
    {
        for (int i = 0; i < seeds.Count; i++) seeds[i].ClusterId = clusterId;
        seeds.Remove(p);
        while (seeds.Count > 0)
        {
            Data currentData = seeds[0];
            List<Data> result = GetRegion(dataList, currentData, eps);
            if (result.Count >= minPts)
            {
                for (int i = 0; i < result.Count; i++)
                {
                    Data resultP = result[i];
                    if (resultP.ClusterId == DataProperties.UNCLUSTERED || resultP.ClusterId == DataProperties.NOISE)
                    {
                        if (resultP.ClusterId == DataProperties.UNCLUSTERED) seeds.Add(resultP);
                        resultP.ClusterId = clusterId;
                    }
                }
            }
            seeds.Remove(currentData);
        }
        return true;
    }
}
```

Code 3.16 Metoda care extinde densitatea

Această metodă face extinderea clusterului sau a densități pe baza numărului minim de exemple(*minPts*). Aceasta apelează metoda *GetRegion* care creează densitatea și returnează o listă de elemente. Această metodă verifică dacă lista returnată are numărul de elemente mai mare sau egal cu *minPts*. După ce s-a creat o regiune se ia fiecare punct din regiunea respectivă și se verifică dacă se poate extinde regiunea, adică se verifică dacă din exemplele de margine ale clusterului se poate face o extindere sau dacă exemplele de margine pot deveni exemple de centru.

Metoda *GetRegion creează* o regiune cu ajutorul matricei de distanțe și a distanței minime între două exemple (*eps*). Această metodă returnează o listă care reprezintă o densitate, o aglomerare a exemplelor.

```
2 references
private List<Data> GetRegion(List<Data> dataList, Data p, double eps)
{
    List<Data> region = new List<Data>();
    //double[] fistData = GetRow(distanceMatrix, p.DocId);

    for (int i = 0; i < dataList.Count; i++)
    {
        //double[] secondData = GetRow(distanceMatrix, dataList[i].DocId);
        double distSquared = distanceMatrix[p.DocId,dataList[i].DocId];

        if (distSquared <= eps) region.Add(dataList[i]);
    }
    return region;
}
```

Code 3.17 Metoda care creează o densitate

3.5.2 HAC (Hierarchical agglomerative clustering)

Acest algoritm face o structură ierarhică a datelor și face gruparea acestora pe baza acestei structuri. Pentru a implementa acest algoritm am creat o clasă *HAC* care conține metode prin care se implementează acest algoritm.

```
2 references
class HAC
{
    private double[,] distanceMatrix;
    1 reference
    public HAC(double[,] distanceMatrix)
    {
        this.distanceMatrix = distanceMatrix;
    }
    1 reference
    public void ImplementationSingleLink(out List<Data> dataList, double eps)...
    1 reference
    public void ImplementationCompleteLink(out List<Data> dataList, double eps)...
    2 references
    private Tuple<int, int, double> Min()...

    2 references
    private Tuple<int, int, double> Max()...
}
```

Code 3.18 Clasa HAC

Această clasă inițializează o matrice care reprezintă matricea de similaritate și pe baza acesteia se clusterizarea exemplelor. În această clasă avem patru metode cu ajutorul cărora se realizează două tipuri de implementare a algoritmului HAC, în funcție de tipul de similaritate ales. La finalul algoritmului se returnează o listă care conține datele grupate.

```
1 reference
public void ImplementationSingleLink(out List<Data> dataList, double eps)
{
    dataList = new List<Data>();//lista care contine id/val liniei de la fiecare linie din mat/document

    //creaza o lista cu toate dic dn matrice si il marcheaza ca fiind cluster unic
    for (var i = 0; i < distanceMatrix.GetLength(0); i++)
    {
        dataList.Add(new Data(i, i));
    }
    Tuple<int, int, double> subCluster = Min();
    double min = subCluster.Item3;
    while (min <= eps)
    {
        if (subCluster.Item1 > subCluster.Item2)
        {
            subCluster = new Tuple<int, int, double>(subCluster.Item2, subCluster.Item1, subCluster.Item3);

        }
        foreach (var item in dataList)
        {
            if (item.ClusterId == subCluster.Item2)
            {
                item.ClusterId = subCluster.Item1;
            }
        }
        for (int i = 0; i < distanceMatrix.GetLength(0); i++)
        {
            if (distanceMatrix[i, subCluster.Item1] > distanceMatrix[i, subCluster.Item2])
            {
                distanceMatrix[i, subCluster.Item1] = distanceMatrix[i, subCluster.Item2];
            }
            distanceMatrix[i, subCluster.Item2] = 9999999999;
        }
        for (int j = 0; j < distanceMatrix.GetLength(1); j++)
        {
            if (distanceMatrix[subCluster.Item1, j] > distanceMatrix[subCluster.Item2, j])
            {
                distanceMatrix[subCluster.Item1, j] = distanceMatrix[subCluster.Item2, j];
            }
            distanceMatrix[subCluster.Item2, j] = 9999999999;
        }
        subCluster = Min();
        min = subCluster.Item3;
    }
}
```

Code 3.19 Metoda care implementează algoritmul HAC SingleLink

Această metodă primește ca parametrii lista care o să conțină, la sfârșit, datele grupate și un *eps* care reprezintă pragul de oprire a dezvoltări structurii ierarhice. Această metodă unește două lini și coloane din matricea de distanțe pe baza valori minime din aceasta. Pentru a uni cele două lini și cele două coloane se calculează și se păstrează valoarea minimă a fiecărui element ale acestora. După ce acestea au fost unite trebuie să se șteargă o linie și o coloană. Ștergerea se face prin înlocuirea tuturor elementelor din linie și coloană cu numărul *9999999999*. Minimul din matrice este calculat cu ajutorul metodei *Min()* care caută și returnează minimul din matrice. Funcția *Min()* returnează și valoarea indexul liniei și coloanei în care minimul a fost găsit.

```
2 references
private Tuple<int, int, double> Min()
{
    double min = Double.MaxValue;
    int indexI = -1;
    int indexJ = -1;
    for (int i = 0; i < distanceMatrix.GetLength(0); i++)
    {
        for (int j = i + 1; j < distanceMatrix.GetLength(1); j++)
        {
            if ((min > distanceMatrix[i, j]))
            {
                min = distanceMatrix[i, j];
                indexI = i;
                indexJ = j;
            }
        }
    }
    return new Tuple<int, int, double>(indexI, indexJ, min);
}
```

Code 3.20 Metoda care caută minimul în matricea de similaritate

```
2 references
private Tuple<int, int, double> Max()
{
    double max = Double.MinValue;
    int indexI = -1;
    int indexJ = -1;
    for (int i = 0; i < distanceMatrix.GetLength(0); i++)
    {
        for (int j = i + 1; j < distanceMatrix.GetLength(1); j++)
        {
            if ((max < distanceMatrix[i, j]))
            {
                max = distanceMatrix[i, j];
                indexI = i;
                indexJ = j;
            }
        }
    }
    return new Tuple<int, int, double>(indexI, indexJ, max);
}
```

Code 3.21 Metoda care caută maximul în matricea de similaritate

```
1 reference
public void ImplementationCompleteLink(out List<Data> dataList, double eps)
{
    dataList = new List<Data>();//lista care contine id/val liniei de la fiecare linie din mat/document

    //creaza o lista cu toate dic dn matrice si il marcheaza ca fiind cluster unic
    for (var i = 0; i < distanceMatrix.GetLength(0); i++)
    {
        dataList.Add(new Data(i, i));
    }
    Tuple<int, int, double> subCluster = Max();
    double max = subCluster.Item3;
    while (max >= eps)
    {
        if (subCluster.Item1 > subCluster.Item2)
        {
            subCluster = new Tuple<int, int, double>(subCluster.Item2, subCluster.Item1, subCluster.Item3);

        }
        foreach (var item in dataList)
        {
            if (item.ClusterId == subCluster.Item2)
            {
                item.ClusterId = subCluster.Item1;
            }
        }
        for (int i = 0; i < distanceMatrix.GetLength(0); i++)
        {
            if ((distanceMatrix[i, subCluster.Item1] < distanceMatrix[i, subCluster.Item2])
               && (distanceMatrix[i, subCluster.Item1] != 0) && (distanceMatrix[i, subCluster.Item2] != 0))
            {
                distanceMatrix[i, subCluster.Item1] = distanceMatrix[i, subCluster.Item2];
            }
            distanceMatrix[i, subCluster.Item2] = -9999999999;
        }
        for (int j = 0; j < distanceMatrix.GetLength(1); j++)
        {
            if ((distanceMatrix[subCluster.Item1, j] < distanceMatrix[subCluster.Item2, j])
                && (distanceMatrix[subCluster.Item1, j] != 0) && (distanceMatrix[subCluster.Item2, j] != 0))
            {
                distanceMatrix[subCluster.Item1, j] = distanceMatrix[subCluster.Item2, j];
            }
            distanceMatrix[subCluster.Item2, j] = -9999999999;
        }
        subCluster = Max();
        max = subCluster.Item3;
    }
}
```

Code 3.22 Metoda care implementează algoritmul HAC CompleteLink

Această metodă este asemănătoare cu metoda anterioară, singura diferență este că aceasta caută maximul din matricea de similaritate cu ajutorul metodei *Max()*. Metoda *Max()* caută și returnează maximul din matricea de similaritate și indexul liniei și coloanei la care acesta a fost găsit. Metoda care implementează tipul CompleteLink șterge linia și coloana prin înlocuirea tuturor elementelor din acestea cu numărul *-9999999999*.

La prima metodă se înlocuiește cu numărul pozitiv, deoarece se caută minimul din matrice și, atunci când dorim să ștergem, trebuie să înlocuim cu un număr foarte mare. La a doua metodă se înlocuiește cu numărul negativ, deoarece se caută minimul din matrice.

3.6 Etapa de atribuire a unei clase pentru un cluster

În această etapă se face corespondența între eticheta datelor și clusteri, adică fiecare cluster o să primească o etichetă și elementele care fac parte din cluster o să primească eticheta respectivă. Această etapă este importantă, deoarece în etapa de evaluare se folosesc metrici externe, care verifică dacă eticheta originală a exemplului este la fel cu eticheta rezultată după aplicarea algoritmilor de clustering.

Un prim pas care este făcut de această etapă este acela de a crea matricea de corespondență. Această matrice conține de câte ori apare fiecare etichetă originală în fiecare cluster, adică câte elemente din interiorul clusterului au avut acele etichete. Aceasta este o matrice de **n × m**, unde „**n**” este numărul de clusteri rezultați și „**m**” este numărul de etichete originale. Fiecare linie reprezintă un cluster, iar fiecare coloană reprezintă o etichetă. Pe fiecare linie din această matrice avem numărul de exemple care au etichetele respective.

```
                Z0 Z3 Z2 Z1
Cluster2: 1024| 0| 0| 0|
Cluster4: 0| 993| 0| 0|
Cluster3: 0| 0| 978| 0|
Cluster1: 0| 0| 0| 994|
```

Fig. 3.4 Matricea de corespondență

```
1 reference
private void CreateCorrespondenceMatrix()
{
    clustersClass = new Dictionary<int, string>();
    Exctragere etichete și clusteri
    correspondenceMatrix = new int[indexClusters.Count, zones.Count];

    //Aici se creaza matricea de corespondenta, se contorizeaza de cate ori apare
    //fiecare clasa in fiecare cluster
    for (var i = 0; i < initialClass.Length; i++)
    {
        correspondenceMatrix[indexClusters.IndexOf(data[i].ClusterId), zones.IndexOf(initialClass[i])]
                    = correspondenceMatrix[indexClusters.IndexOf(data[i].ClusterId), zones.IndexOf(initialClass[i])] + 1;
    }

    if (indexClusters.Count>(zones.Count+1))
    {
        isNoise = true;
    }
    classCombinatiaMaxima = new CombinatiaMaxima(ref correspondenceMatrix);
    combinatiaMaxima = classCombinatiaMaxima.GetCombinatieMaxima();

    foreach (var elem in combinatiaMaxima)
    {
        clustersClass.Add(indexClusters[elem.mIndexLinie], zones[elem.mIndexColoana]);
    }
}
```

Code 3.23 Metoda care creează matricea de corespondență

```
#region Exctragere etichete și clusteri
zones = new List<string>();
for (var i = 0; i < initialClass.Length; i++)
{
    if (!zones.Contains(initialClass[i]))
    {
        zones.Add(initialClass[i]);
    }
}
indexClusters = new List<int>();
foreach (var d in data)
{
    if (!indexClusters.Contains(d.ClusterId))
    {
        indexClusters.Add(d.ClusterId);
    }
}
#endregion
```

Code 3.24 Salvare etichete și clusteri

Această metodă creează matricea de corespondentă și atribuie fiecărui cluster o etichetă. La început trebuie să fie salvate etichetele originale și clusteri rezultați pentru a putea contoriza de câte ori a apărut o etichetă în fiecare cluster. După ce acestea au fost salvate se parcurge vectorul de etichete rezultat după etapa de citire a datelor și se creează matrice de corespondență. Următorul pas atribuie fiecărui cluster o etichetă. Această atribuire se face cu metoda *GetCombinatieMaxima()* din clasa *CombinatiaMaxima*. Această metodă atribuie fiecărui cluster eticheta, pe baza matricei de corespondență. Această metodă implementează un algoritm de backtracking care determină combinația care are valoarea maximă din matrice și pe baza acestui maxim se atribuie fiecărui cluster o etichetă.

```
1 reference
public List<Element> GetCombinatieMaxima()
{
    BackTracking(0);
    Console.WriteLine("Maxim: " + valMaxim);
    foreach (var elem in combinatiaMaxim)
    {
        Console.WriteLine(elem.mIndexLinie.ToString() +
            " " + elem.mIndexColoana.ToString() + " " + elem.mValoare.ToString());
    }
    return combinatiaMaxim;
}
```

Code 3.25 Metoda care returnează combinația maximă

```
2 references
private void BackTracking(byte nivel)
{

    if ((nivel == nrLine) || (combinatiaCurenta.Count() == nrCol))
    {
        int tempMaxim = 0;
        foreach (var elem in combinatiaCurenta)
        {
            tempMaxim += elem.mValoare;
        }
        if (tempMaxim > valMaxim)
        {
            valMaxim = tempMaxim;
            combinatiaMaxim.Clear();
            foreach (var elem in combinatiaCurenta)
                combinatiaMaxim.Add(elem);
        }
        return;
    }
    List<byte> intermediar = new List<byte>();
    bool salt;
    for (byte i = 0; i < nrCol; i++)
    {
        salt = false;
        foreach (var elem in combinatiaCurenta)
        {
            if (elem.mIndexColoana == i)
            {
                salt = true;
                break;
            }
        }
        if (!salt)
        {
            if (correspondenceMatrix[nivel, i] != 0)
                intermediar.Add(i);
        }
    }
    Element e;
    foreach (var elem in intermediar)
    {
        e = new Element(nivel, elem, (Int16)correspondenceMatrix[nivel, elem]);
        combinatiaCurenta.Add(e);
        if (nivel < nrLine) BackTracking((byte)(nivel + 1));
        combinatiaCurenta.Remove(e);
    }
    return;
}
```

Code 3.26 Metoda care calculează combinația care are valoarea maximă

Următorul pas al acestei etape este acela de a eticheta toate exemplele care au fost folosite în etapa de clustering cu o etichetă nouă. Pentru a face acest lucru trebuie să calculăm medoizi tuturor clusterilor, iar după să se calculeze distanța între fiecare exemplu și toți medoizi. Fiecare exemplu o să primească eticheta clusterului care are distanța între exemplu și medoid cea mai mică, adică eticheta clusterului care este cel mai similar.

```
1 reference
private Dictionary<int, int> GetMedoizi()
{
    Dictionary<int, int> returnCent = new Dictionary<int, int>();


    for (int k = 0; k < indexClusters.Count; k++)
    {
        if (indexClusters[k] != -1)
        {
            List<int> docInCluster = new List<int>();
            double Min = Double.MaxValue;
            int indexCentru = 0;
            for (int i = 0; i < data.Count; i++)
            {
                if (data[i].ClusterId == indexClusters[k])
                {
                    docInCluster.Add(data[i].DocId);
                }
            }

            for (int i = 0; i < docInCluster.Count; i++)
            {
                double sum = 0;
                for (int j = 0; j < docInCluster.Count; j++)
                {
                    sum += distanceMatrix[docInCluster[i], docInCluster[j]];
                }
                if (sum < Min)
                {
                    indexCentru = docInCluster[i];
                    Min = sum;
                }
            }
            returnCent.Add(indexClusters[k], indexCentru);
        }
    }

    return returnCent;
}
```

Code 3.27 Metoda care calculează medoidul din fiecare cluster

Pentru a determina medoidul unui cluster, se face suma distanțelor dintre fiecare exemplu din cluster și toate celelatle exemple care aparțin clusterului și se ia ca medoid exemplul cu suma distanțelor cea mai mică.

```
1 reference
private void DocPerClass()
{
    docPerClass = new List<string>();
    medoizi = GetMedoizi();


    for (int i = 0; i < data.Count; i++)
    {
        List<double> distanteDocCentroid = new List<double>();
        List<int> centroidDoc = new List<int>();
        foreach (var item in medoizi)
        {
            distanteDocCentroid.Add(distanceMatrix[item.Value, data[i].DocId]);
            centroidDoc.Add(item.Value);
        }

        double minDist = distanteDocCentroid.Min();
        int doc = centroidDoc[distanteDocCentroid.IndexOf(minDist)];
        if (isNoise)
        {

            if (clustersClass.ContainsKey(data[doc].ClusterId))
            {
                docPerClass.Add(clustersClass[data[doc].ClusterId]);
            }
            else
            {
                docPerClass.Add("noise");
            }
        }
        else
        {
            docPerClass.Add(clustersClass[data[doc].ClusterId]);
        }

    }

}
```

Code 3.28 Metoda care atribuie fiecărui punct eticheta nouă

Această metodă parcurge toate exemplele și pe baza distanței între fiecare exemplu și medoizi se etichetează fiecare exemplu. Se calculează distanța între fiecare exemplu și toți medoizi calculați anterior, iar eticheta nouă a exemplului o să fie eticheta clusterului care are medoidul cel mai apropiat de acesta.

3.7 Etapa de evaluare

Această etapă face evaluarea procesului de clustering prin intermediul meticilor externe. În ceea ce urmează am să prezint pașii necesari pentru a calcula metricile externe și cum sunt aceștia implementați.

Primul pas al acestei etape este acela de a calcula matricea de eroare, deoarece pe baza acesteia se calculează metricile externe.

```
1 reference
private void CreateConfusionMatrix()
{
    if (isNoise)
    {
        zones.Add("noise");
    }

    confusionMatrix = new int[zones.Count, zones.Count];

    for (int i = 0; i < initialClass.Length; i++)
    {
        confusionMatrix[zones.IndexOf(docPerClass[i]), zones.IndexOf(initialClass[i])]++;

    }
}
```

Code 3.29 Metoda care calculează matricea de eroare

Această metodă creează matricea de eroare. Începe prin inițializarea unei matrici de **n × n**, unde „**n**” reprezintă numărul de etichete. Se parcurg datele și se incrementează valoarea găsită la indexul oferit de vechea și noua etichetă, adică se adună cu 1 valoarea din matrice, de la indexul etichetei vechi și de la indexul celei noi.

După ce se creează această matrice trebuie să se calculeze true positive, true negative, false positive și false negative. Acestea sunt calculate pe baza matricei calculate anterior pentru fiecare etichetă în parte și sunt salvate într-o listă în care un element conține cele patru caracteristici pentru o etichetă.

Următorul pas după ce se calculează true positive, true negative, false positive și false negative este acela de a calcula metricile externe cu ajutorul acestora. La final, după ce s-au calculat metricile externe folosite, trebuie să face o analiză pe baza acelor metrici cu ajutorul căreia se determină care algoritm a fost mai eficient și se poate determina și ce implementare a fiecărei etape este mai bună pentru fiecare algoritm.

```
1 reference
private void CalculMetrici()
{
    List<ConfusionMatrixData> confusionMatrixDatas = new List<ConfusionMatrixData>();
    metriciExterne = new MetriciExterne();

    for (int k = 0; k < zones.Count; k++)
    {
        int tP = confusionMatrix[k, k];
        int tN = 0;
        int fP = 0;
        int fN = 0;
        for (int i = 0; i < confusionMatrix.GetLength(0); i++)
        {
            for (int j = 0; j < confusionMatrix.GetLength(1); j++)
            {
                if (i != k && j != k)
                {
                    tN += confusionMatrix[i, j];
                }
                else if (i == k && j != k)
                {
                    fP += confusionMatrix[i, j];
                }

                else if (i != k && j == k)
                {
                    fN += confusionMatrix[i, j];
                }
            }

        }
        confusionMatrixDatas.Add(new ConfusionMatrixData(tP, tN, fP, fN));
    }
    metriciExterne.Calc(ref confusionMatrixDatas, isNoise);
}
```

Code 3.30 Metoda care calculează metricile externe

Această metodă calculează metricile externe pentru fiecare etichetă și media aritmetică pentru fiecare metrică. Această metodă calculează true positive, true negative, false positive și false negative pentru fiecare etichetă în parte pe baza matricei de eroare. Acestea sunt salvate într-o listă de tipul *ConfusionMatrixData*. Matricea de eroare este parcursă pentru fiecare etichetă în parte și la finalul unei parcurgeri se salvează cele patru informații în lista, iar indexul listei de tipul *ConfusionMatrixData* este același cu indexul listei *zones*. Lista *zones* conține etichetele, iar prin folosirea aceluiași index pentru cele două liste cunoaștem true positive, true negative, false positive și false negative pentru fiecare etichetă în parte. Pentru a se putea calcula metricile externe trebuie să se inițializeze un obiect de tipul *MetriciExterne* și să apelăm metoda de calcul din această clasă.

Clasa *MetriciExterne* calculeaza metricile externe pentru fiecare etichetă în parte cât și o medie aritmetică a fiecărei metrici. Metricile sunt salvate într-o listă care conține liste cu cele trei metricile, astfel fiecare element din listă conține o altă listă de tipul double de trei elemente. Astfel

salvăm metricile pentru fiecare etichetă în parte. Medie aritmetică pentru fiecare metrică este salvată într-o variabilă de tipul double care are numele metrici respective (*double accuracy*, *double precision*, *double recall*).

```
4 references
public class ConfusionMatrixData
{
    private int tP;
    private int tN;
    private int fP;
    private int fN;

    1 reference
    public ConfusionMatrixData(int tP, int tN, int fP, int fN)
    {
        this.tP = tP;
        this.tN = tN;
        this.fP = fP;
        this.fN = fN;
    }

    16 references
    public int TP { get => tP; set => tP = value; }
    4 references
    public int TN { get => tN; set => tN = value; }
    6 references
    public int FP { get => fP; set => fP = value; }
    6 references
    public int FN { get => fN; set => fN = value; }
}
```

Code 3.31 Clasa *ConfusionMatrixData*

```
7 references
public class MetriciExterne
{
    List<List<double>> metriciPerClass;
    double accuracy;
    double precision;
    double recall;


    1 reference
    public MetriciExterne()
    {
        this.accuracy = 0;
        this.precision = 0;
        this.recall = 0;
    }
    1 reference
    public void Calc(ref List<ConfusionMatrixData> confusionMatrixDatas,bool isNoise)...
    3 references
    public double Accuracy { get => accuracy; set => accuracy = value; }
    3 references
    public double Precision { get => precision; set => precision = value; }
    3 references
    public double Recall { get => recall; set => recall = value; }
    3 references
    public List<List<double>> MetriciPerClass { get => metriciPerClass; set => metriciPerClass = value; }
}
```

Code 3.32 Clasa MetriciExterne

```
1 reference
public void Calc(ref List<ConfusionMatrixData> confusionMatrixDatas,bool isNoise)
{
    metriciPerClass = new List<List<double>>();
    int index = 0;
    if (!isNoise)
    {
        foreach (var elem in confusionMatrixDatas)
        {
            metriciPerClass.Add(new List<double>());
            double tempAcc = Convert.ToDouble(elem.TP + elem.TN) / Convert.ToDouble(elem.TP + elem.TN + elem.FP + elem.FN);
            double tempPrec = 0;
            double tempRec = 0;

            //Verificam daca numitorul este 0
            if (Convert.ToDouble(elem.TP + elem.FP) != 0)
            {
                tempPrec = Convert.ToDouble(elem.TP) / Convert.ToDouble(elem.TP + elem.FP);
            }
            else
            {
                tempPrec = 0;
            }

            //Verificam daca numitorul este 0
            if (Convert.ToDouble(elem.TP + elem.FN) != 0)
            {
                tempRec = Convert.ToDouble(elem.TP) / Convert.ToDouble(elem.TP + elem.FN);
            }
            else
            {
                tempPrec = 0;
            }
            accuracy += tempAcc;
            precision += tempPrec;
            recall += tempRec;

            metriciPerClass[index].Add(tempAcc);
            metriciPerClass[index].Add(tempPrec);
            metriciPerClass[index].Add(tempRec);
            index++;
        }

        accuracy = accuracy / Convert.ToDouble(index);
        precision = precision / Convert.ToDouble(index);
        recall = recall / Convert.ToDouble(index);
    }
```

Code 3.33 Metoda care calculează metricile externe, implementare fără zgomot

Această metodă calculează cele trei metrici externe. Există două tipuri de implementări, cu si fără zgomot, deoarece zgomotul nu poate fi considerat relevant. Metricile trebuie să ia zgomotul în considerare din punct de vedere statistic. Metoda primește ca parametrii lista care conține true positive, true negative, false positive și false negative pentru fiecare etichetă și o variabilă (*isNoise*) prin care știm dacă avem sau nu zgomot. Cu ajutorul acestei variabile se alege tipul de implementare, cu sau fără zgomot. Această metodă returnează lista care conține metricile tuturor etichetelor și media aritmetică pentru fiecare metrică.

```
    else
    {
        for (int i = 0; i < confusionMatrixDatas.Count-1; i++)
        {
            metriciPerClass.Add(new List<double>());
            double tempAcc = Convert.ToDouble(confusionMatrixDatas[i].TP + confusionMatrixDatas[i].TN) /
                Convert.ToDouble(confusionMatrixDatas[i].TP + confusionMatrixDatas[i].TN + confusionMatrixDatas[i].FP +
                confusionMatrixDatas[i].FN);
            double tempPrec = 0;
            double tempRec = 0;

            //Verificam daca numitorul este 0
            if (Convert.ToDouble(confusionMatrixDatas[i].TP + confusionMatrixDatas[i].FP) != 0)
            {
                tempPrec = Convert.ToDouble(confusionMatrixDatas[i].TP) /
                    Convert.ToDouble(confusionMatrixDatas[i].TP + confusionMatrixDatas[i].FP);
            }
            else
            {
                tempPrec = 0;
            }

            //Verificam daca numitorul este 0
            if (Convert.ToDouble(confusionMatrixDatas[i].TP + confusionMatrixDatas[i].FN) != 0)
            {
                tempRec = Convert.ToDouble(confusionMatrixDatas[i].TP) /
                    Convert.ToDouble(confusionMatrixDatas[i].TP + confusionMatrixDatas[i].FN);
            }
            else
            {
                tempPrec = 0;
            }
            accuracy += tempAcc;
            precision += tempPrec;
            recall += tempRec;

            metriciPerClass[index].Add(tempAcc);
            metriciPerClass[index].Add(tempPrec);
            metriciPerClass[index].Add(tempRec);
            index++;
        }

        accuracy = accuracy / Convert.ToDouble(index);
        precision = precision / Convert.ToDouble(index);
        recall = recall / Convert.ToDouble(index);
    }
}
```

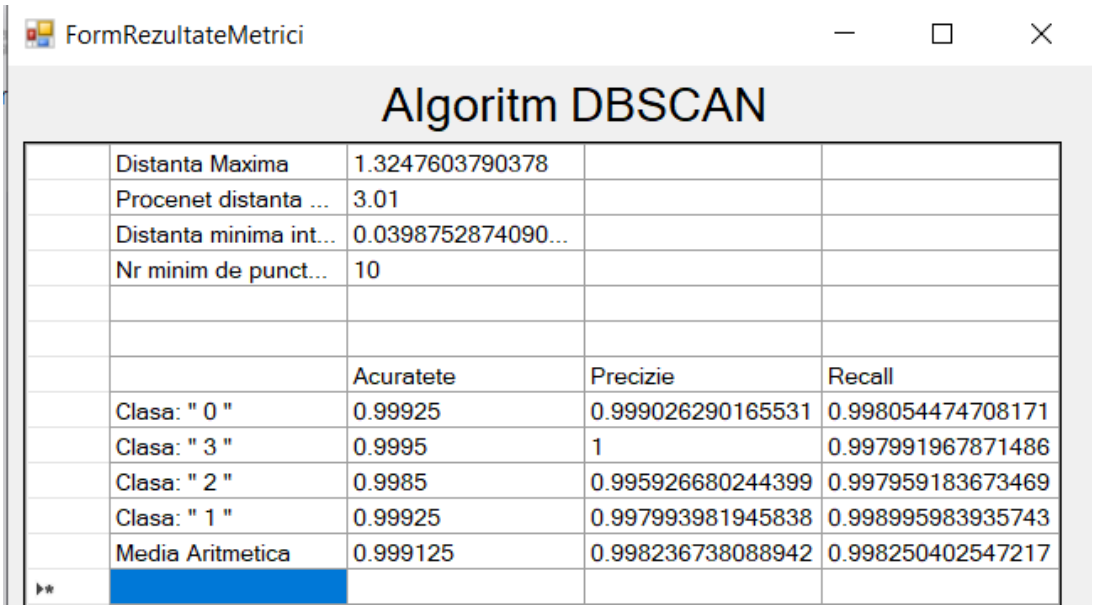
Code 3.34 Metoda care calculează metricile externe, implementare cu zgomot

3.8 Rezultate experimentale

În acest subcapitol am să prezint câteva rezultate obținute după aplicarea procesului de clustering. Am să prezint rezultate atât după aplicarea procesului pe fișiere care conțin puncte în spațiul 2D, cît și după aplicarea proesului pe fișiere care conțin documente sub formă de vectori.

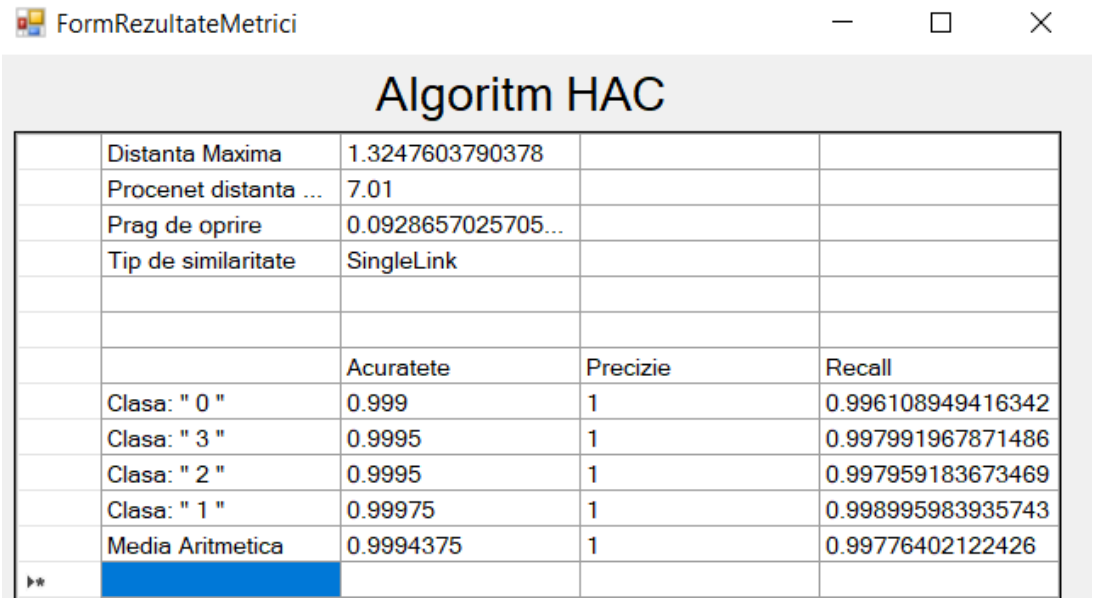
1. Fișiere care conțin puncte 2D

- Rezultate obținute după aplicarea normalizări Min-Max și calculul similarității cu distanța euclidiană.



	Distanta Maxima	1.3247603790378		
	Procenet distanta ...	3.01		
	Distanta minima int...	0.0398752874090...		
	Nr minim de punct...	10		
		Acuratete	Precizie	Recall
	Clasa: " 0 "	0.99925	0.999026290165531	0.998054474708171
	Clasa: " 3 "	0.9995	1	0.997991967871486
	Clasa: " 2 "	0.9985	0.995926680244399	0.997959183673469
	Clasa: " 1 "	0.99925	0.997993981945838	0.998995983935743
	Media Aritmetica	0.999125	0.998236738088942	0.998250402547217
▸*				

Fig. 3.5 Rezultate DBSCAN cu normalizare Min-Max și distanța euclidiană



	Distanta Maxima	1.3247603790378		
	Procenet distanta ...	7.01		
	Prag de oprire	0.0928657025705...		
	Tip de similaritate	SingleLink		
		Acuratete	Precizie	Recall
	Clasa: " 0 "	0.999	1	0.996108949416342
	Clasa: " 3 "	0.9995	1	0.997991967871486
	Clasa: " 2 "	0.9995	1	0.997959183673469
	Clasa: " 1 "	0.99975	1	0.998995983935743
	Media Aritmetica	0.9994375	1	0.99776402122426
▸*				

Fig. 3.6 Rezultate HAC cu normalizare Min-Max și distanța euclidiană

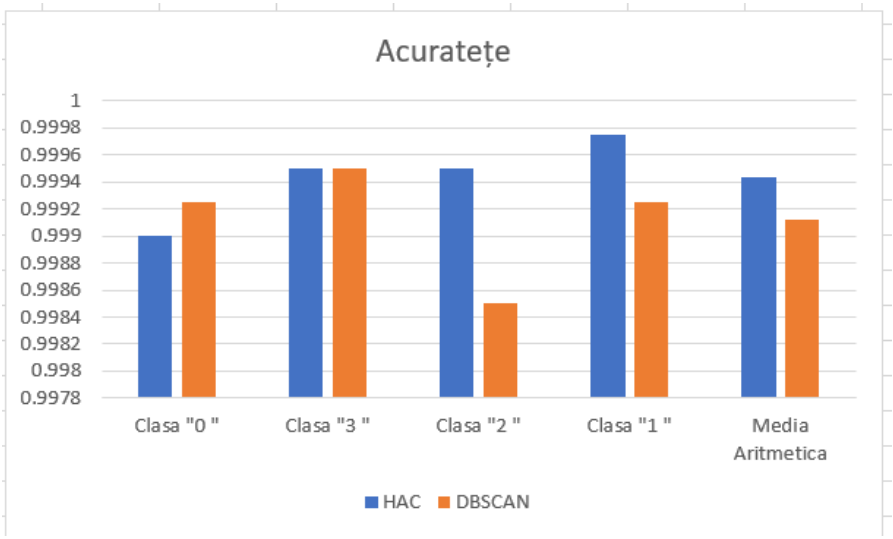


Fig. 3.7 Comparație acuratețe cu normalizarea Min-Max și distanța euclidiană

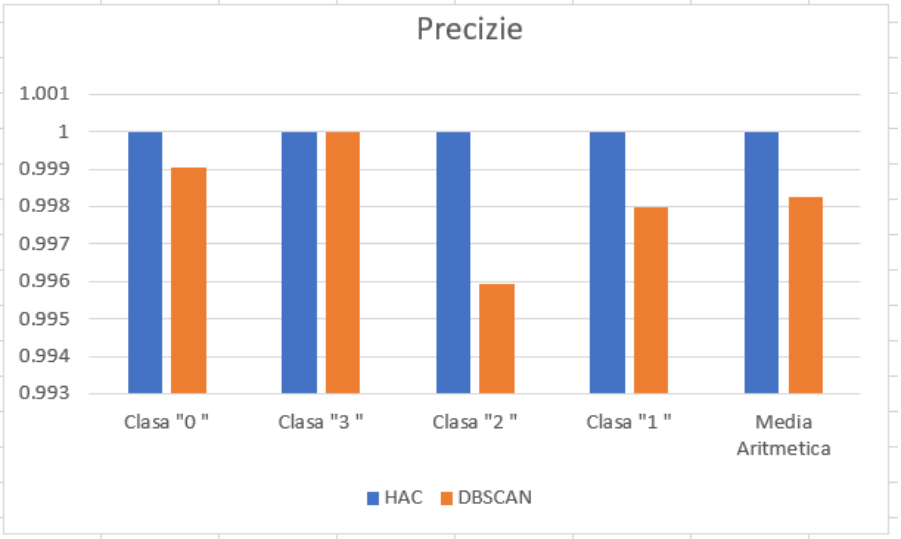


Fig. 3.8 Comparație precizie cu normalizarea Min-Max și distanța euclidiană

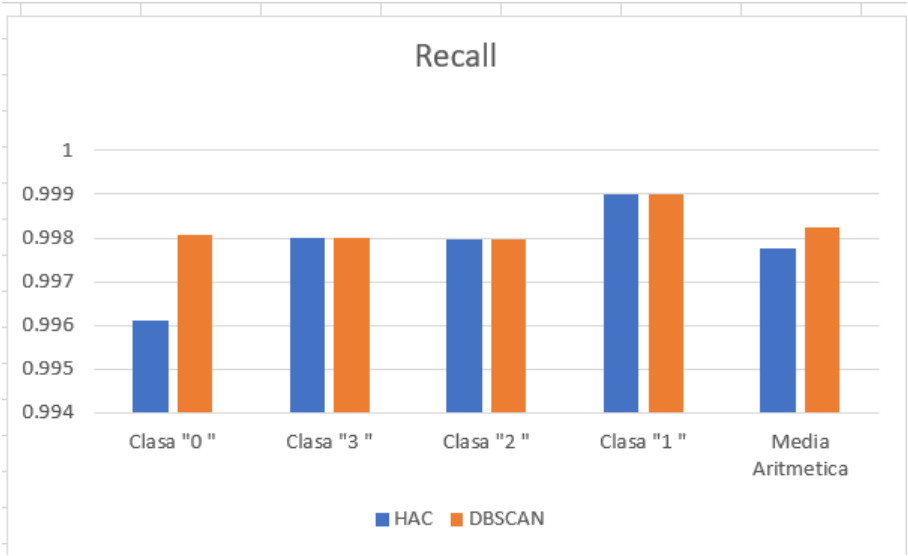
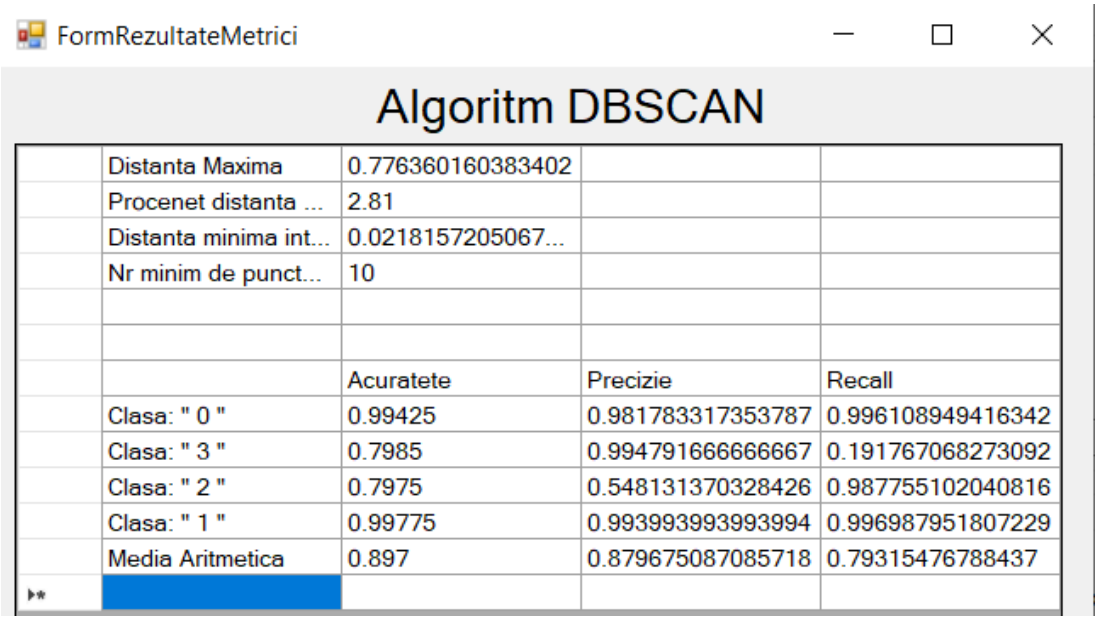


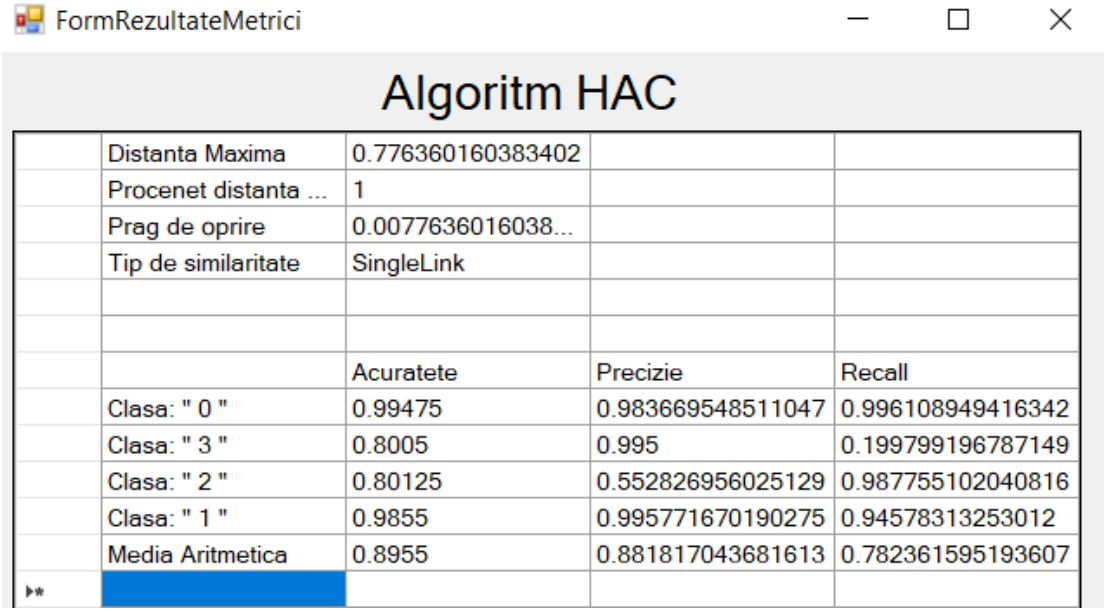
Fig. 3.9 Comparație recall cu normalizarea Min-Max și distanța euclidiană

- Rezultate obținute după aplicarea normalizări Suma1 și calculul similarității cu distanța euclidiană.



	Distanta Maxima	0.776360160383402		
	Procenet distanta ...	2.81		
	Distanta minima int...	0.0218157205067...		
	Nr minim de punct...	10		
		Acuratete	Precizie	Recall
	Clasa: " 0 "	0.99425	0.981783317353787	0.996108949416342
	Clasa: " 3 "	0.7985	0.994791666666667	0.191767068273092
	Clasa: " 2 "	0.7975	0.548131370328426	0.987755102040816
	Clasa: " 1 "	0.99775	0.993993993993994	0.996987951807229
	Media Aritmetica	0.897	0.879675087085718	0.79315476788437
▶*				

Fig. 3.10 Rezultate DBSCAN cu normalizare Suma1 și distanța euclidiană



	Distanta Maxima	0.776360160383402		
	Procenet distanta ...	1		
	Prag de oprire	0.0077636016038...		
	Tip de similaritate	SingleLink		
		Acuratete	Precizie	Recall
	Clasa: " 0 "	0.99475	0.983669548511047	0.996108949416342
	Clasa: " 3 "	0.8005	0.995	0.199799196787149
	Clasa: " 2 "	0.80125	0.552826956025129	0.987755102040816
	Clasa: " 1 "	0.9855	0.995771670190275	0.94578313253012
	Media Aritmetica	0.8955	0.881817043681613	0.782361595193607
▶*				

Fig. 3.11 Rezultate HAC cu normalizare Suma1 și distanța euclidiană

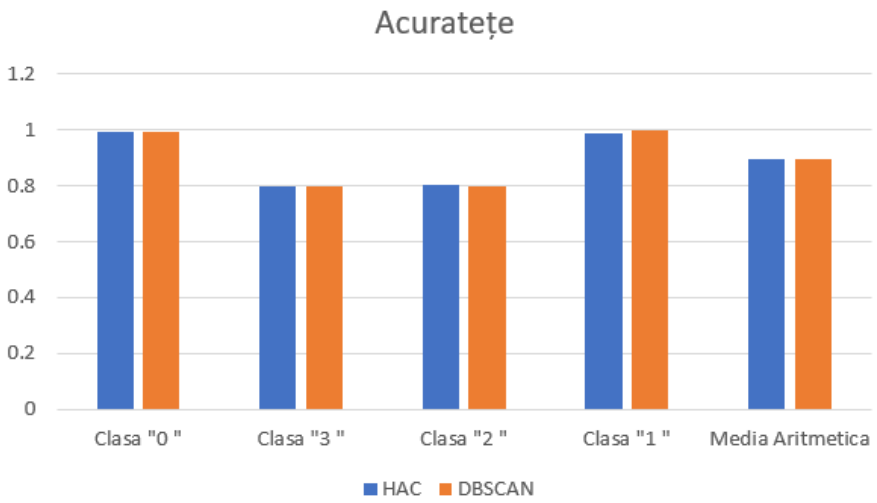


Fig. 3.12 Comparație acuratețe cu normalizarea Suma1 și distanța euclidiană

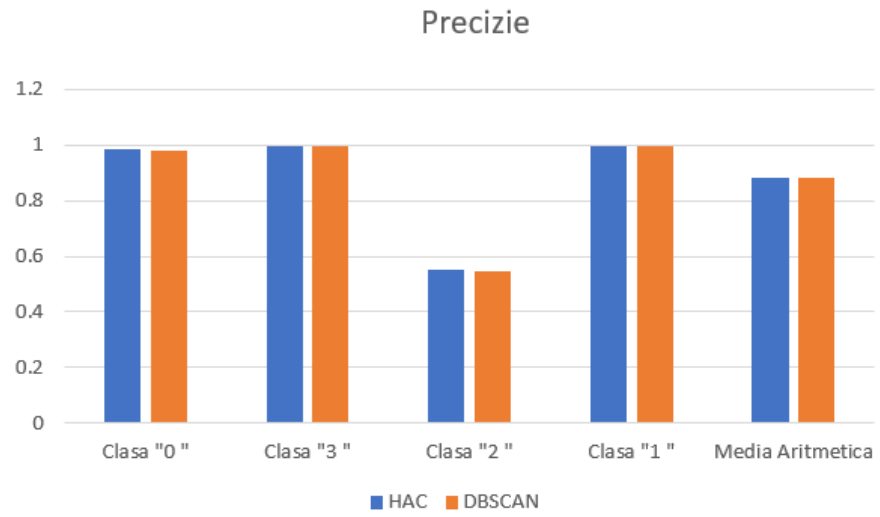


Fig. 3.13 Comparație precizie cu normalizarea Suma1 și distanța euclidiană

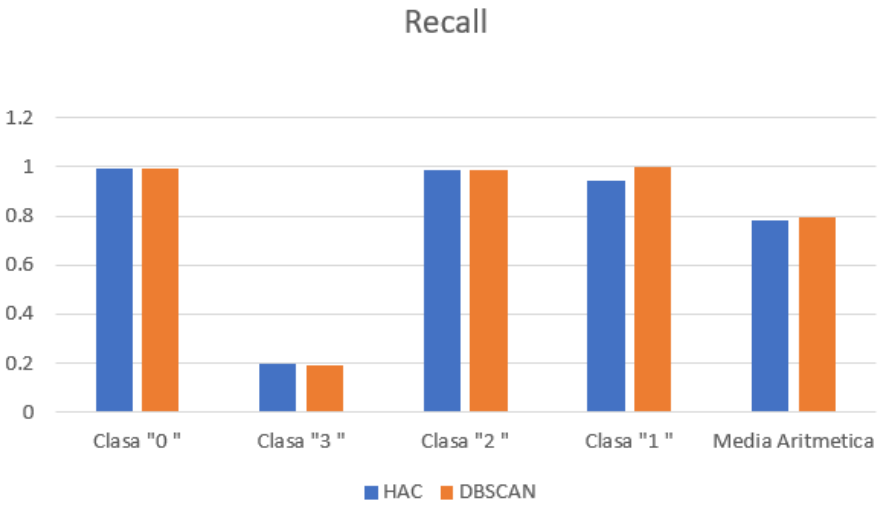
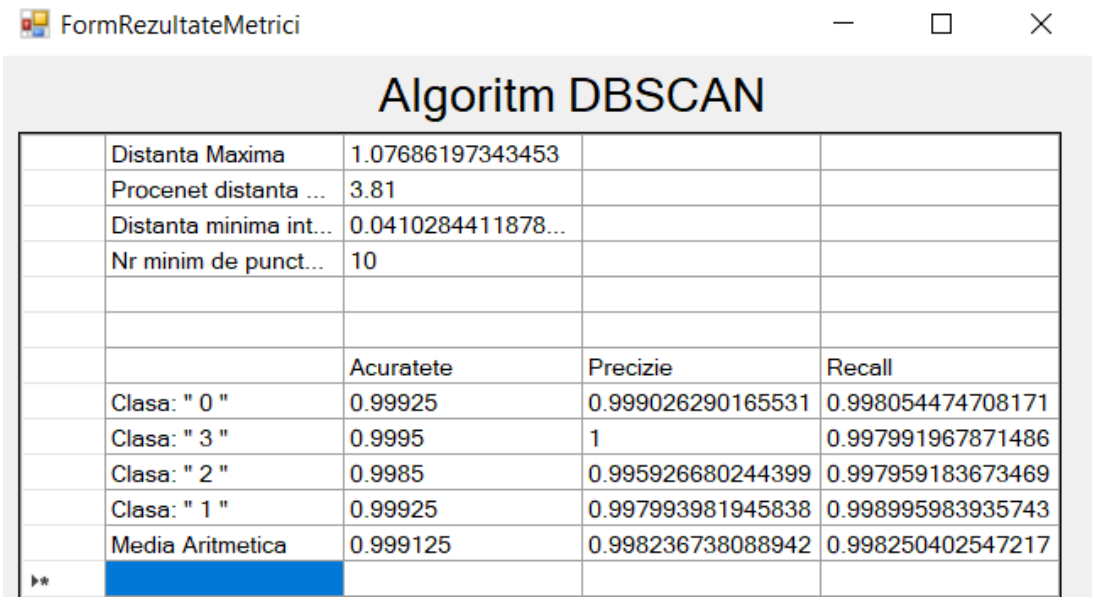


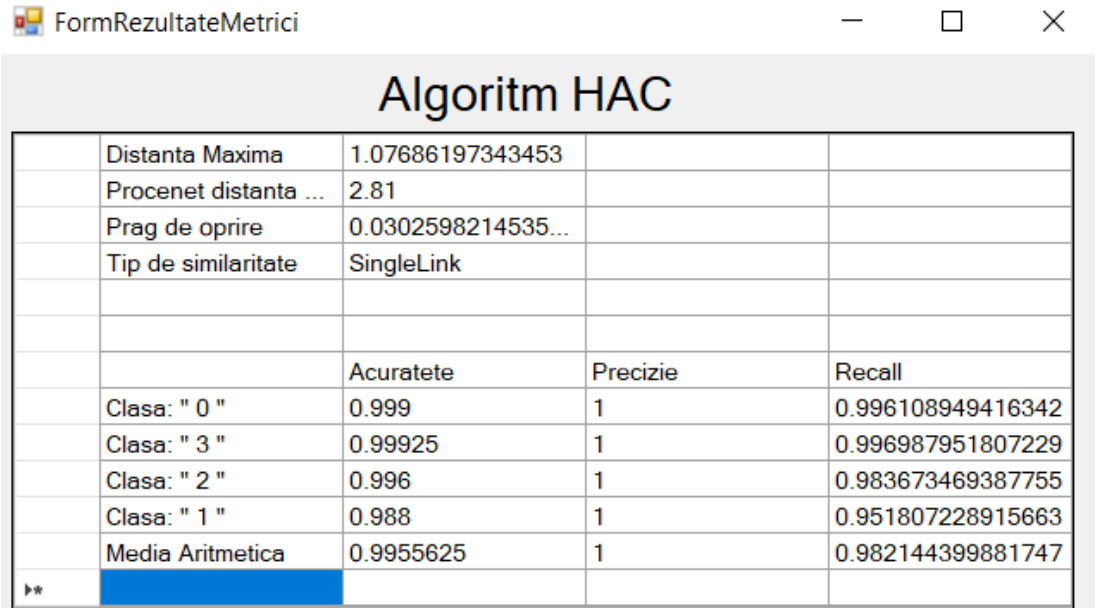
Fig. 3.14 Comparație recall cu normalizarea Suma1 și distanța euclidiană

- Rezultate obținute după aplicarea normalizări Min-Max și calculul similarității cu distanța Minkowski.



	Distanta Maxima	1.07686197343453		
	Procenet distanta ...	3.81		
	Distanta minima int...	0.0410284411878...		
	Nr minim de punct...	10		
		Acuratete	Precizie	Recall
	Clasa: " 0 "	0.99925	0.999026290165531	0.998054474708171
	Clasa: " 3 "	0.9995	1	0.997991967871486
	Clasa: " 2 "	0.9985	0.995926680244399	0.997959183673469
	Clasa: " 1 "	0.99925	0.997993981945838	0.998995983935743
	Media Aritmetica	0.999125	0.998236738088942	0.998250402547217
▸*				

Fig. 3.15 Rezultate DBSCAN cu normalizare Min-Max și distanța Minkowski



	Distanta Maxima	1.07686197343453		
	Procenet distanta ...	2.81		
	Prag de oprire	0.0302598214535...		
	Tip de similaritate	SingleLink		
		Acuratete	Precizie	Recall
	Clasa: " 0 "	0.999	1	0.996108949416342
	Clasa: " 3 "	0.99925	1	0.996987951807229
	Clasa: " 2 "	0.996	1	0.983673469387755
	Clasa: " 1 "	0.988	1	0.951807228915663
	Media Aritmetica	0.9955625	1	0.982144399881747
▸*				

Fig. 3.16 Rezultate HAC cu normalizare Min-Max și distanța Minkowski

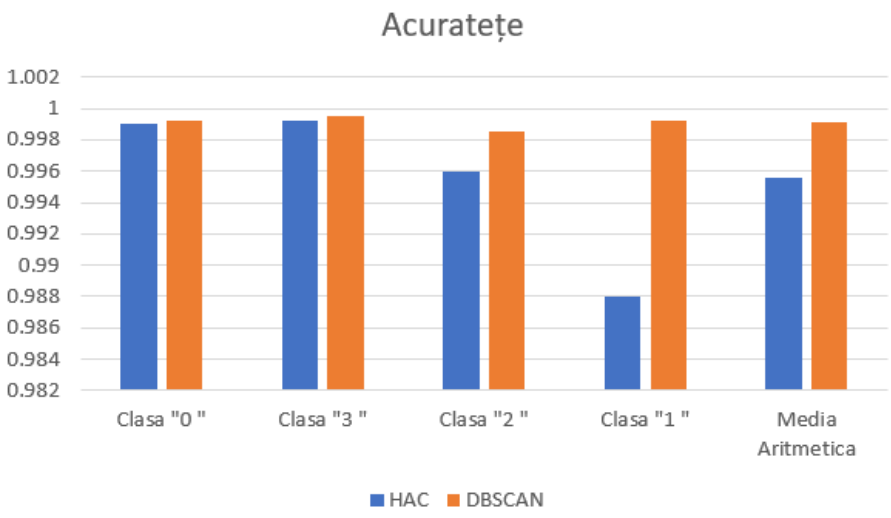


Fig. 3.17 Comparație acuratete cu normalizarea Min-Max și distanța Minkowski

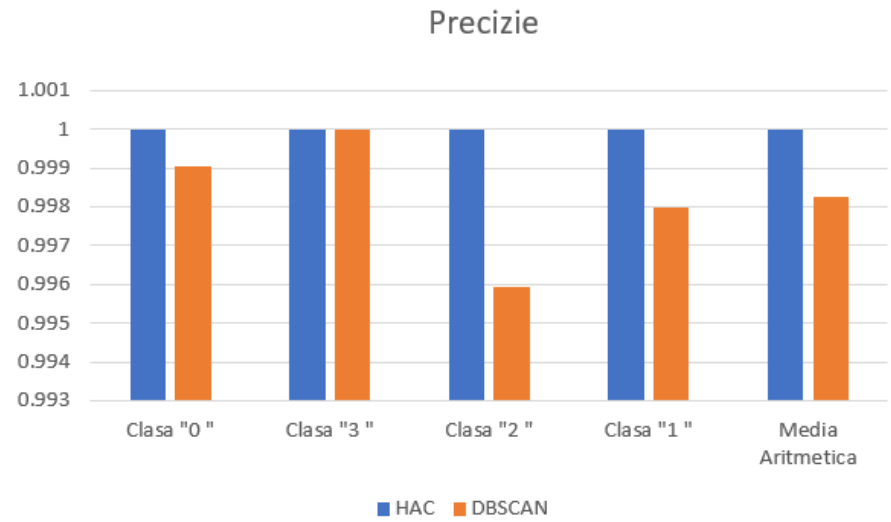


Fig. 3.18 Comparație precizie cu normalizarea Min-Max și distanța Minkowski

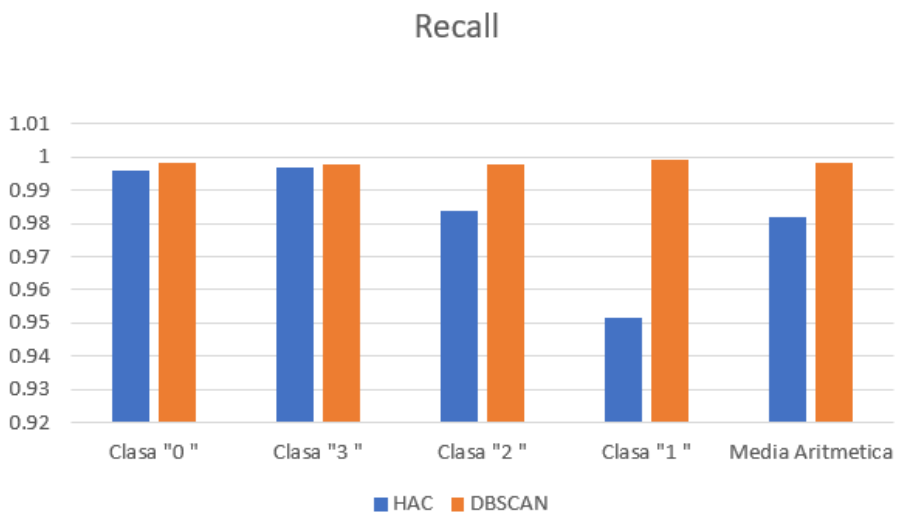
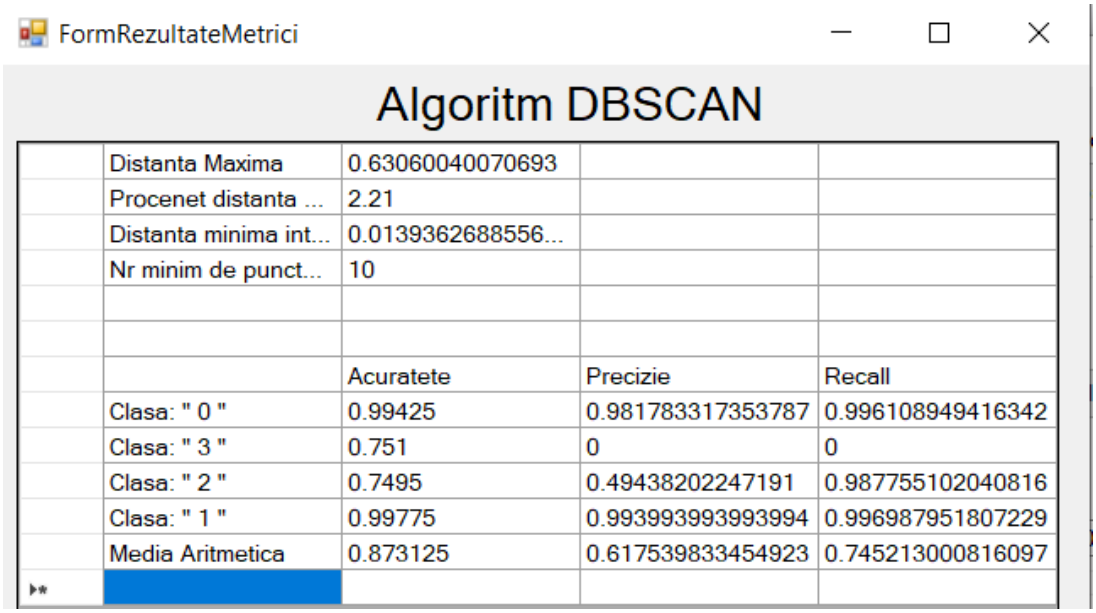


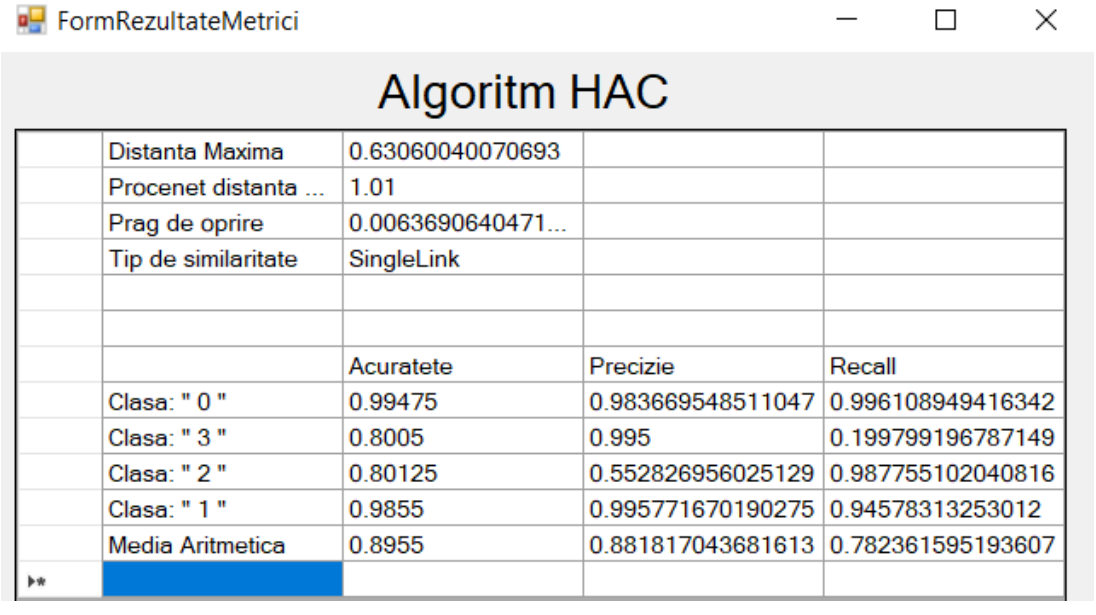
Fig. 3.19 Comparație recall cu normalizarea Min-Max și distanța Minkowski

- Rezultate obținute după aplicarea normalizări Suma1 și calculul similarității cu distanța Minkowski.



	Distanta Maxima	0.63060040070693		
	Procenet distanta ...	2.21		
	Distanta minima int...	0.0139362688556...		
	Nr minim de punct...	10		
		Acuratete	Precizie	Recall
	Clasa: " 0 "	0.99425	0.981783317353787	0.996108949416342
	Clasa: " 3 "	0.751	0	0
	Clasa: " 2 "	0.7495	0.49438202247191	0.987755102040816
	Clasa: " 1 "	0.99775	0.993993993993994	0.996987951807229
	Media Aritmetica	0.873125	0.617539833454923	0.745213000816097
▸*				

Fig. 3.20 Rezultate DBSCAN cu normalizare Suma1 și distanța Minkowski



	Distanta Maxima	0.63060040070693		
	Procenet distanta ...	1.01		
	Prag de oprire	0.0063690640471...		
	Tip de similaritate	SingleLink		
		Acuratete	Precizie	Recall
	Clasa: " 0 "	0.99475	0.983669548511047	0.996108949416342
	Clasa: " 3 "	0.8005	0.995	0.199799196787149
	Clasa: " 2 "	0.80125	0.552826956025129	0.987755102040816
	Clasa: " 1 "	0.9855	0.995771670190275	0.94578313253012
	Media Aritmetica	0.8955	0.881817043681613	0.782361595193607
▸*				

Fig. 3.21 Rezultate HAC cu normalizare Suma1 și distanța Minkowski

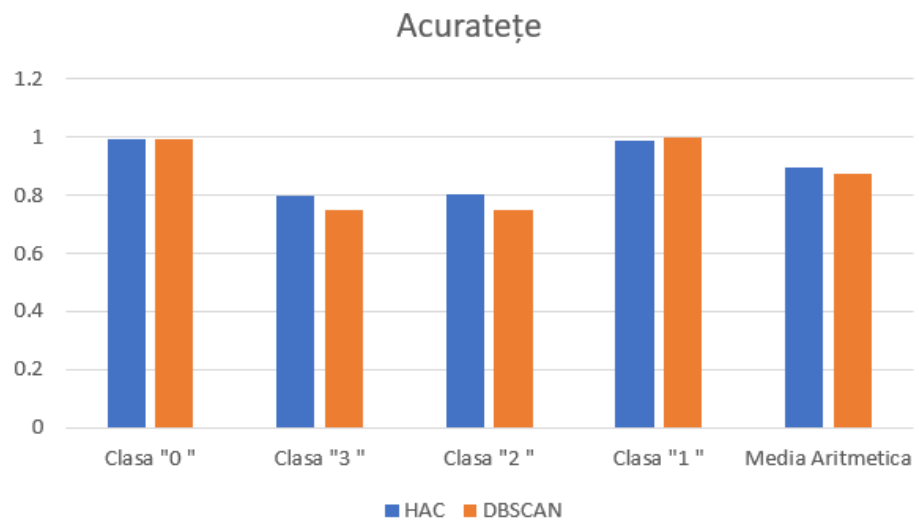


Fig. 3.22 Comparație acuratețecu normalizarea Suma1 și distanța Minkowski

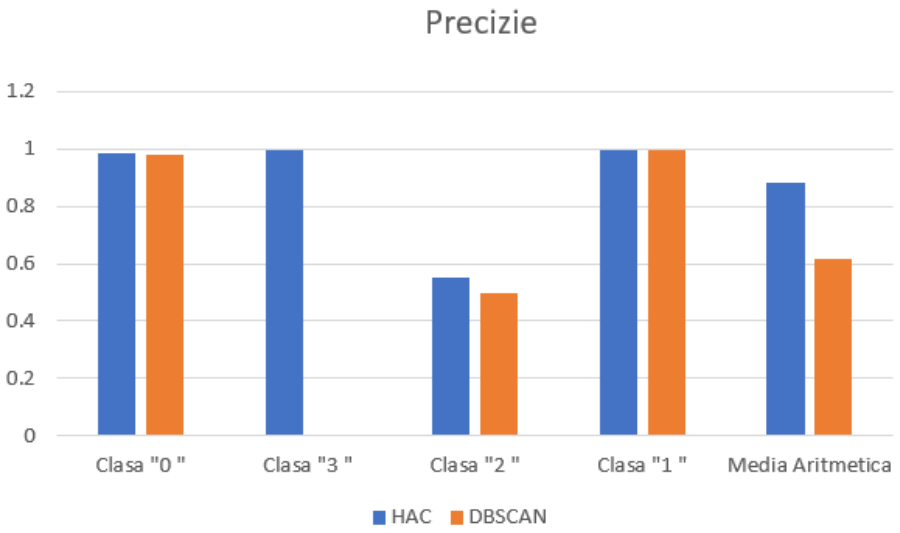


Fig. 3.23 Comparație precizie cu normalizarea Suma1 și distanța Minkowski

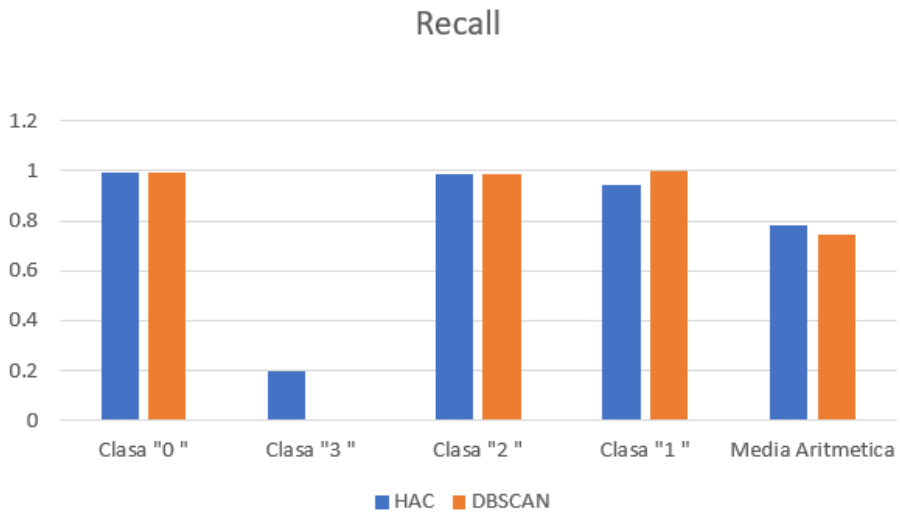
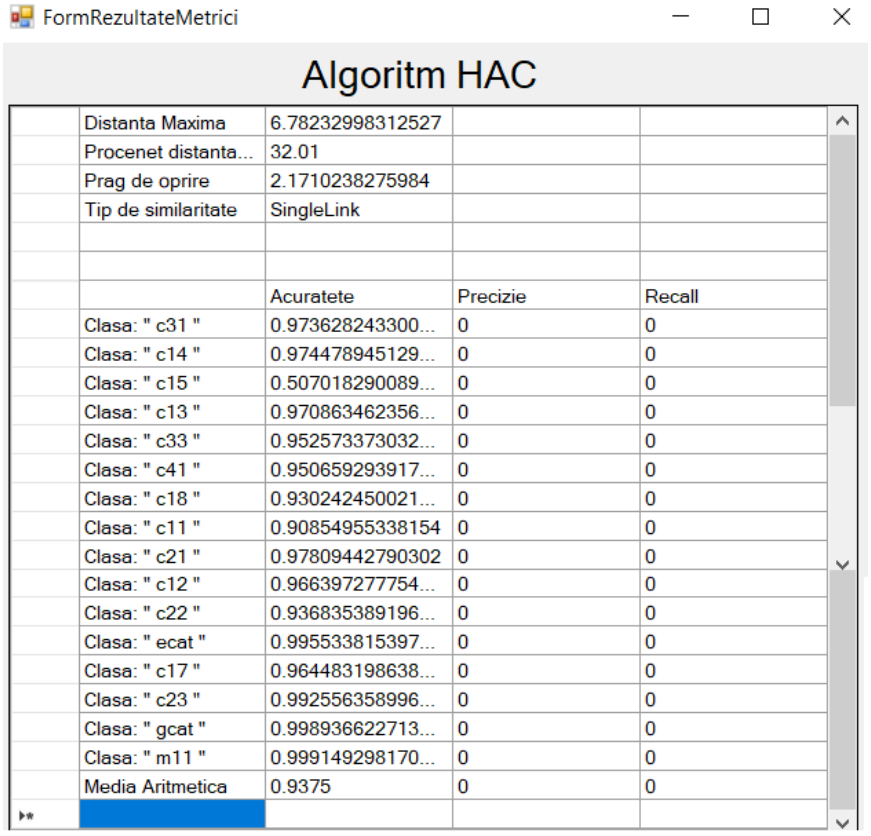


Fig. 3.24 Comparație recall cu normalizarea Suma1 și distanța Minkowski

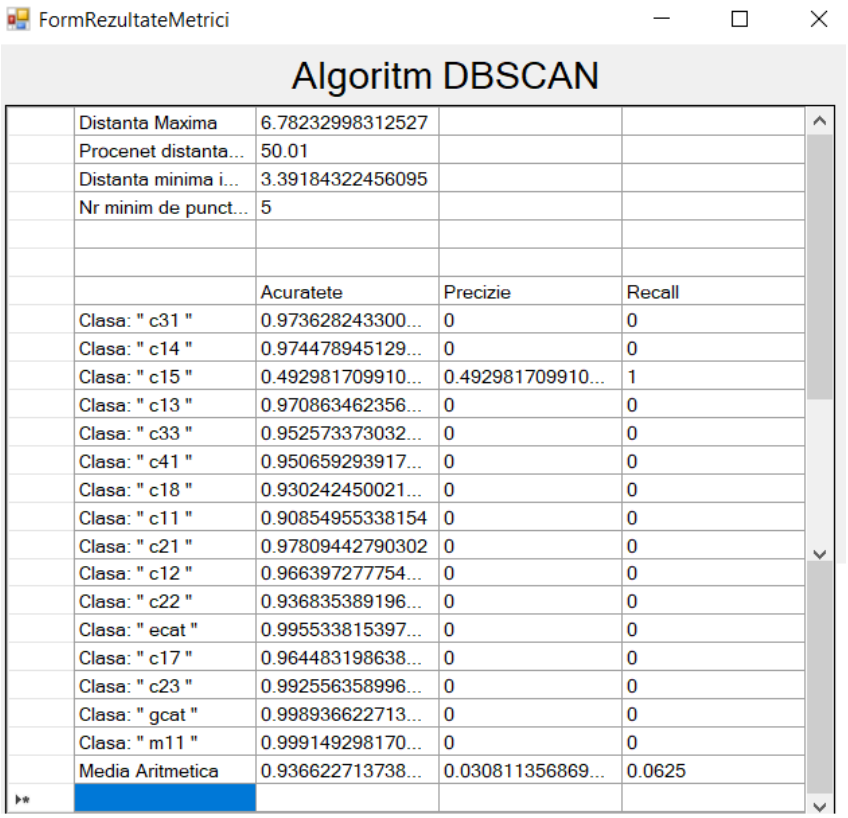
2. Fișiere care conțin documente sub formă de vectori

- Rezultate obținute după aplicarea normalizări Nominale și calculul distanței euclidiane.



	Distanta Maxima	6.78232998312527		
	Procenet distanta...	32.01		
	Prag de oprire	2.1710238275984		
	Tip de similaritate	SingleLink		
		Acuratete	Precizie	Recall
	Clasa: " c31 "	0.973628243300...	0	0
	Clasa: " c14 "	0.974478945129...	0	0
	Clasa: " c15 "	0.507018290089...	0	0
	Clasa: " c13 "	0.970863462356...	0	0
	Clasa: " c33 "	0.952573373032...	0	0
	Clasa: " c41 "	0.950659293917...	0	0
	Clasa: " c18 "	0.930242450021...	0	0
	Clasa: " c11 "	0.90854955338154	0	0
	Clasa: " c21 "	0.97809442790302	0	0
	Clasa: " c12 "	0.966397277754...	0	0
	Clasa: " c22 "	0.936835389196...	0	0
	Clasa: " ecat "	0.995533815397...	0	0
	Clasa: " c17 "	0.964483198638...	0	0
	Clasa: " c23 "	0.992556358996...	0	0
	Clasa: " gcat "	0.998936622713...	0	0
	Clasa: " m11 "	0.999149298170...	0	0
	Media Aritmetica	0.9375	0	0

Fig. 3.25 Rezultate HAC cu normalizare Nominală și distanța euclidiană



	Distanta Maxima	6.78232998312527		
	Procenet distanta...	50.01		
	Distanta minima i...	3.39184322456095		
	Nr minim de punct...	5		
		Acuratete	Precizie	Recall
	Clasa: " c31 "	0.973628243300...	0	0
	Clasa: " c14 "	0.974478945129...	0	0
	Clasa: " c15 "	0.492981709910...	0.492981709910...	1
	Clasa: " c13 "	0.970863462356...	0	0
	Clasa: " c33 "	0.952573373032...	0	0
	Clasa: " c41 "	0.950659293917...	0	0
	Clasa: " c18 "	0.930242450021...	0	0
	Clasa: " c11 "	0.90854955338154	0	0
	Clasa: " c21 "	0.97809442790302	0	0
	Clasa: " c12 "	0.966397277754...	0	0
	Clasa: " c22 "	0.936835389196...	0	0
	Clasa: " ecat "	0.995533815397...	0	0
	Clasa: " c17 "	0.964483198638...	0	0
	Clasa: " c23 "	0.992556358996...	0	0
	Clasa: " gcat "	0.998936622713...	0	0
	Clasa: " m11 "	0.999149298170...	0	0
	Media Aritmetica	0.936622713738...	0.030811356869...	0.0625

Fig. 3.26 Rezultate DBSCAN cu normalizare Nominală și distanța euclidiană

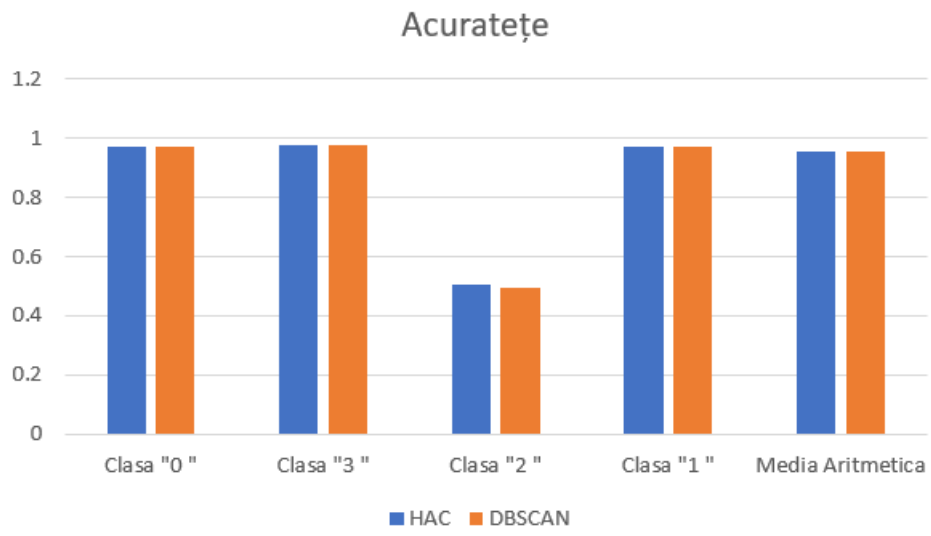


Fig. 3.27 Comparație acuratețe cu normalizarea Nominală și distanța euclidiană

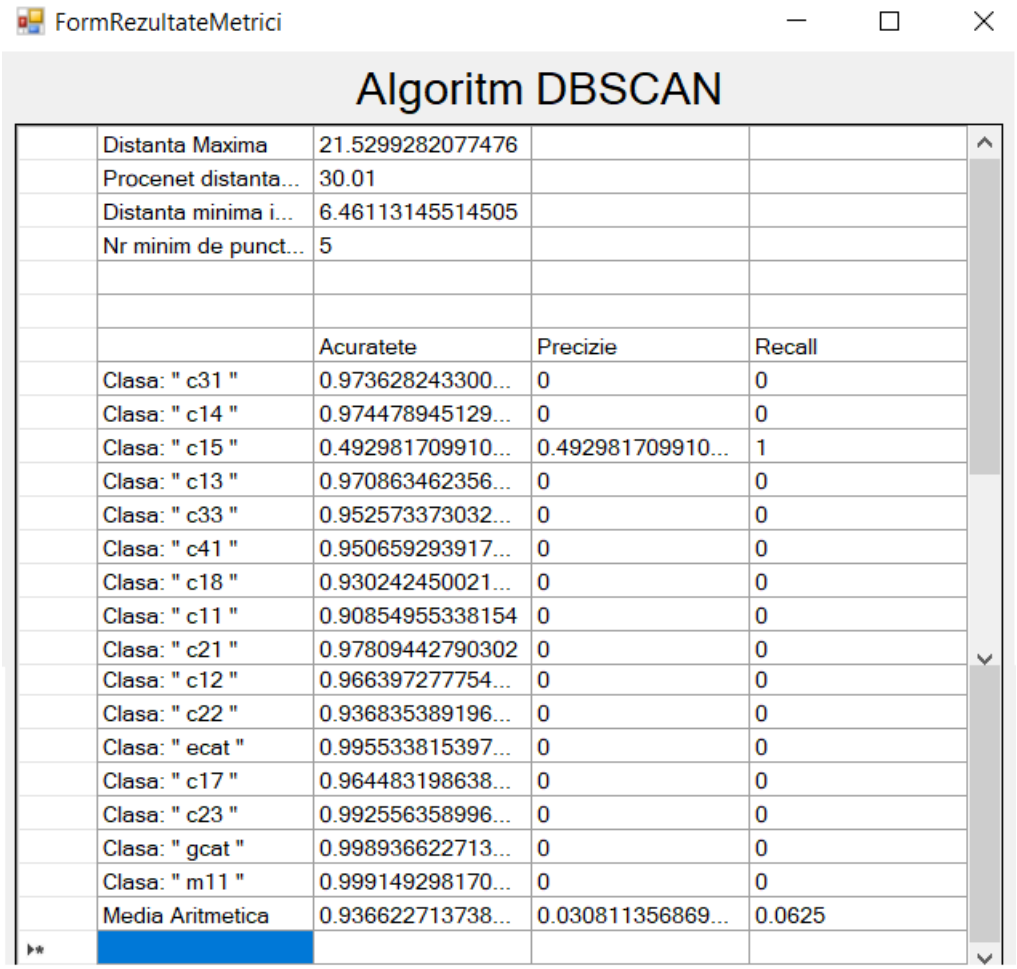
- Rezultate obținute după aplicarea normalizări Cornell-Smart și calculul distanței euclidiane.

FormRezultateMetrici

Algoritm HAC

	Distanta Maxima	21.5299282077476		
	Procenet distanta...	32.01		
	Prag de oprire	6.89173001930001		
	Tip de similaritate	SingleLink		
		Acuratete	Precizie	Recall
	Clasa: " c31 "	0.973628243300...	0	0
	Clasa: " c14 "	0.974478945129...	0	0
	Clasa: " c15 "	0.507018290089...	0	0
	Clasa: " c13 "	0.970863462356...	0	0
	Clasa: " c33 "	0.952573373032...	0	0
	Clasa: " c41 "	0.950659293917...	0	0
	Clasa: " c18 "	0.930242450021...	0	0
	Clasa: " c11 "	0.90854955338154	0	0
	Clasa: " c21 "	0.97809442790302	0	0
	Clasa: " c12 "	0.966397277754...	0	0
	Clasa: " c22 "	0.936835389196...	0	0
	Clasa: " ecat "	0.995533815397...	0	0
	Clasa: " c17 "	0.964483198638...	0	0
	Clasa: " c23 "	0.992556358996...	0	0
	Clasa: " gcat "	0.998936622713...	0	0
	Clasa: " m11 "	0.999149298170...	0	0
	Media Aritmetica	0.9375	0	0
▸*				

Fig. 3.28 Rezultate HAC cu normalizare Cornell-Smart și distanța euclidiană



	Distanta Maxima	21.5299282077476		
	Procenet distanta...	30.01		
	Distanta minima i...	6.46113145514505		
	Nr minim de punct...	5		
		Acuratete	Precizie	Recall
	Clasa: " c31 "	0.973628243300...	0	0
	Clasa: " c14 "	0.974478945129...	0	0
	Clasa: " c15 "	0.492981709910...	0.492981709910...	1
	Clasa: " c13 "	0.970863462356...	0	0
	Clasa: " c33 "	0.952573373032...	0	0
	Clasa: " c41 "	0.950659293917...	0	0
	Clasa: " c18 "	0.930242450021...	0	0
	Clasa: " c11 "	0.90854955338154	0	0
	Clasa: " c21 "	0.97809442790302	0	0
	Clasa: " c12 "	0.966397277754...	0	0
	Clasa: " c22 "	0.936835389196...	0	0
	Clasa: " ecat "	0.995533815397...	0	0
	Clasa: " c17 "	0.964483198638...	0	0
	Clasa: " c23 "	0.992556358996...	0	0
	Clasa: " gcat "	0.998936622713...	0	0
	Clasa: " m11 "	0.999149298170...	0	0
	Media Aritmetica	0.936622713738...	0.030811356869...	0.0625
▶*				

Fig. 3.29 Rezultate DBSCAN cu normalizare Cornell-Smart și distanța euclidiană

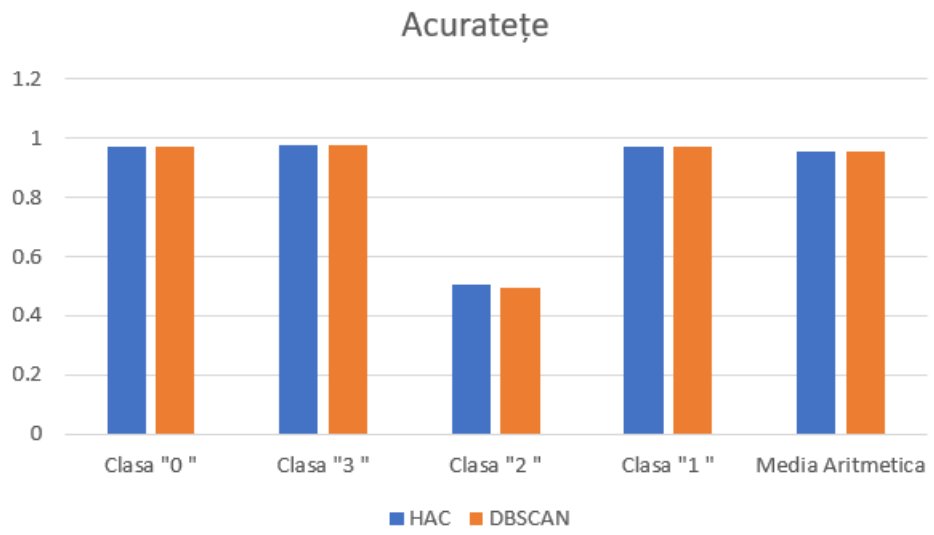


Fig. 3.30 Comparație acurateţe cu normalizarea Cornell-Smart și distanța euclidiană

Se poate observa ca pe documente avem aceleași rezultate atât cu normalizarea Nominală cât și cu normalizarea Cornell-Smart. Comparația între algoritmi poate fi făcută doar pe acurateţe, deoarece precizia si recall-ul la algoritmul HAC sunt 0.

3.9 Interfața aplicației

Pentru a face aplicația care implementează procesul de clustering am folosit limbajul C#. Am creat un proiect de tip Windows Form Application pentru a crea interfața aplicației.

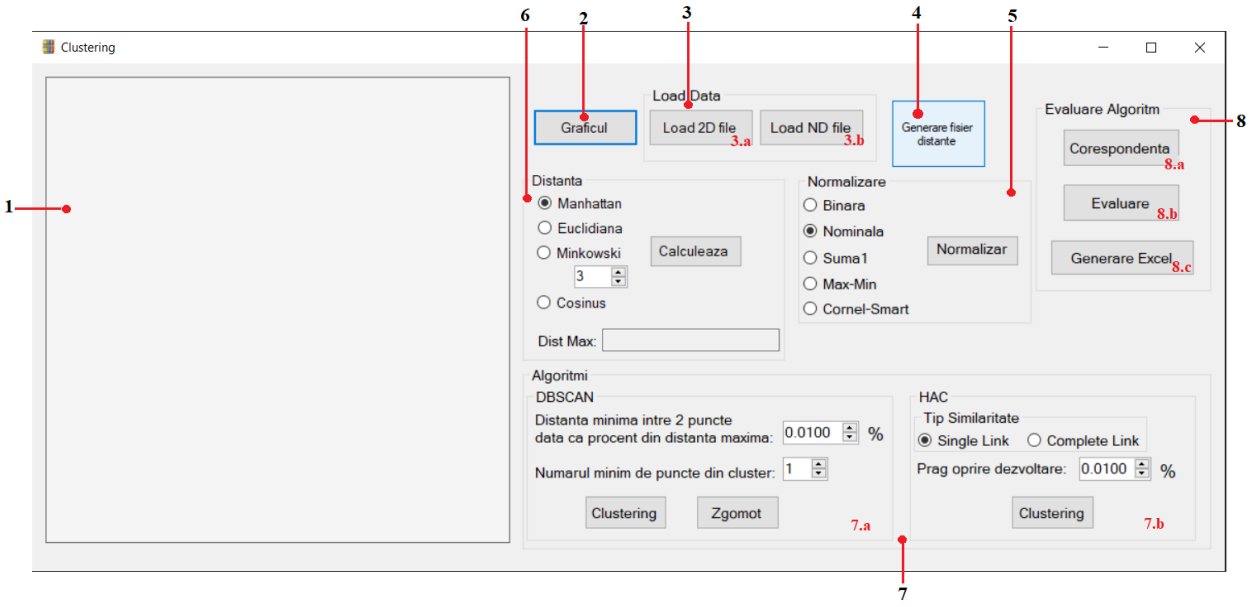


Fig. 3.31 Interfața aplicației

Aceasta este interfața aplicației care creează procesul de clustering și face evaluarea acestuia. În ceea ce urmează am să prezint elementele de pe interfață și cum se folosesc acestea.

1. Acesta este un panou în care sunt afișate punctele care trebuie clusterizate atunci când se lucrează cu puncte. În acest panou sunt afișate punctele înainte de clustering și punctele după clustering, fiecare având culoarea clusterului din care face parte.
2. Acest buton afișează pe panoul **1**, axele și limitele graficului din care fac parte punctele.

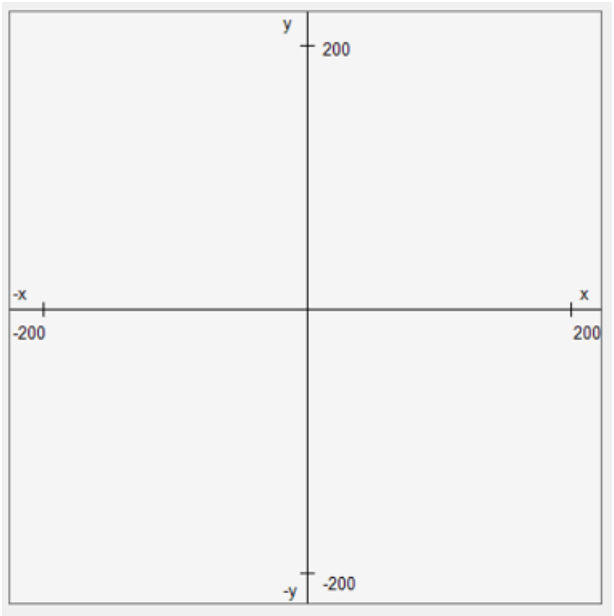


Fig. 3.32 Panoul 1 care afișează axele și limitele graficului

3. Aceasta este o zonă care conține două butoane și care implementează etapa de citire a datelor. Butonul *Load 2D file* deschide o altă fereastră de căutare de unde putem alege fișierul pe care vrem să îl citim. După ce este ales fișierul și este citit punctele sunt afișate în Panoul **1**. Butonul *Load ND file* face citirea fișierelor cu extensia **.arff** care conțin vectori care reprezintă documente. Și aici se deschide o nouă fereastră care ne ajută să alegem fișierul pe care dorim să îl citim.

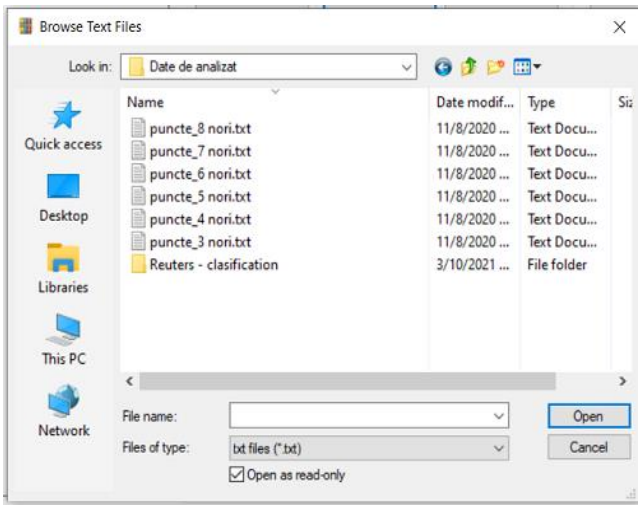


Fig. 3.33 Fereastra pentru căutarea fișierului

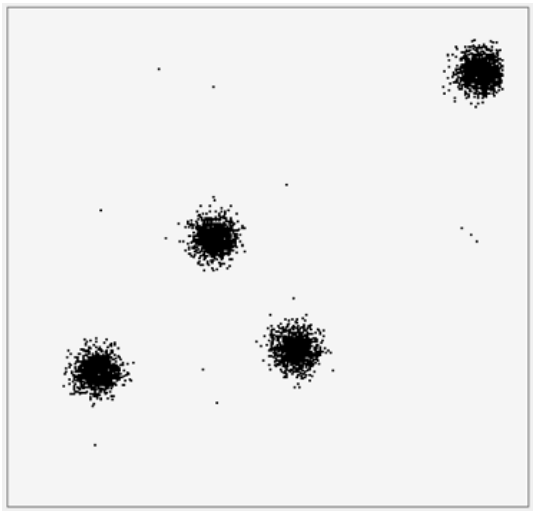


Fig. 3.34 Punctele afișate după citire

4. Acest buton generează un fișier text care conține distanțele din matricea de distanțe în ordine crescătoare. Acesta se poate folosi doar după ce se calculează matricea de distanțe.
5. Aceasta este o zonă care implementează etapa de normalizare a datelor. Aici se alege tipul de normalizare pe care dorim să îl folosim și se apasă pe butonul *Normalizare*.

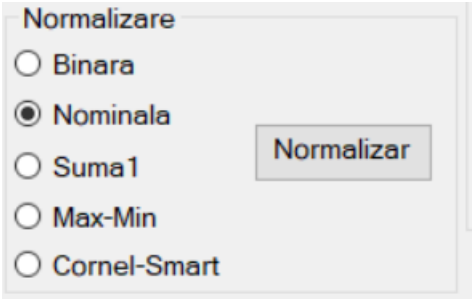


Fig. 3.35 Interfața pentru normalizarea datelor

6. Aceasta este o zonă care implementează etapa de calcul a matricei de similaritate. De aici se alege tipul de distanță pe care dorim să îl folosim pentru calculul similarității și se apasă pe butonul *Calculeaza*. Aici se afișează și distanța maximă pe baza căreia se calculează pragul pentru algoritmul HAC și distanța minimă între două puncte la algoritmul DBSCAN.

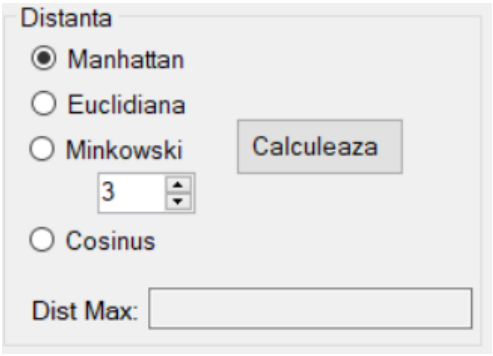


Fig. 3.36 Interfața pentru calculul matricei de similaritate

7. În această zonă se face implementarea celor doi algoritmi de clustering. Aici se găsesc alte două zone care implementează fiecare algoritm și în care se introduc parametrii de intrare pentru fiecare algoritm. Zona *DBSCAN* aplică algoritmul DBSCAN după apăsarea butonului *Clustering* din interiorul zonei. În această zonă putem introduce parametrii cu care lucrează algoritmul. La finalul aplicării algoritmului pe puncte se afișează punctele în panoul **1** cu o culoare care caracterizează clusterul respectiv. În cazul punctelor putem afișa și punctele grupate ca zgomot prin apăsarea butonului *Zgomot*. Punctele grupate ca zgomot sunt afișate cu roșu. Distanța minimă între două puncte este dată ca procent din distanța maximă calculată anterior.

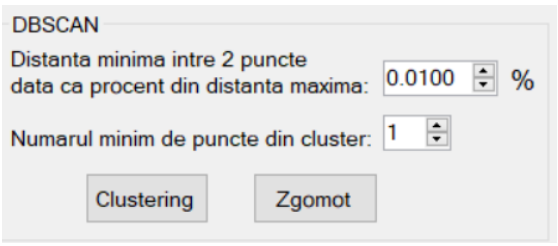


Fig. 3.37 Interfața DBSCAN

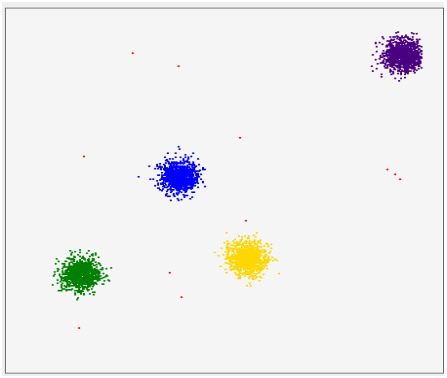


Fig. 3.38 Afișarea punctelor după clustering

Zona care aplică algoritmul HAC ne oferă posibilitatea de a alege tipul de similaritate folosit și valoarea de prag, dată ca procent din distanța maximă.

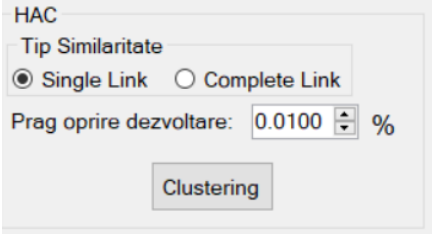


Fig. 3.39 Interfața HAC

8. Această zonă implementează două etape, etapa de atribuire a unei clase pentru un cluster și etapa de evaluare. La apăsarea butonului *Corespondenta* se calculează matricea de corespondență și se face atribuirea unei etichete fiecărui cluste. În această etapă se afișează un mesaj care conține metricea de corespondență.

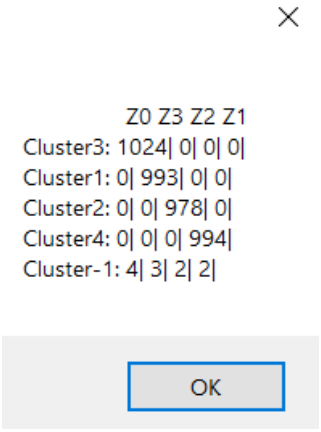
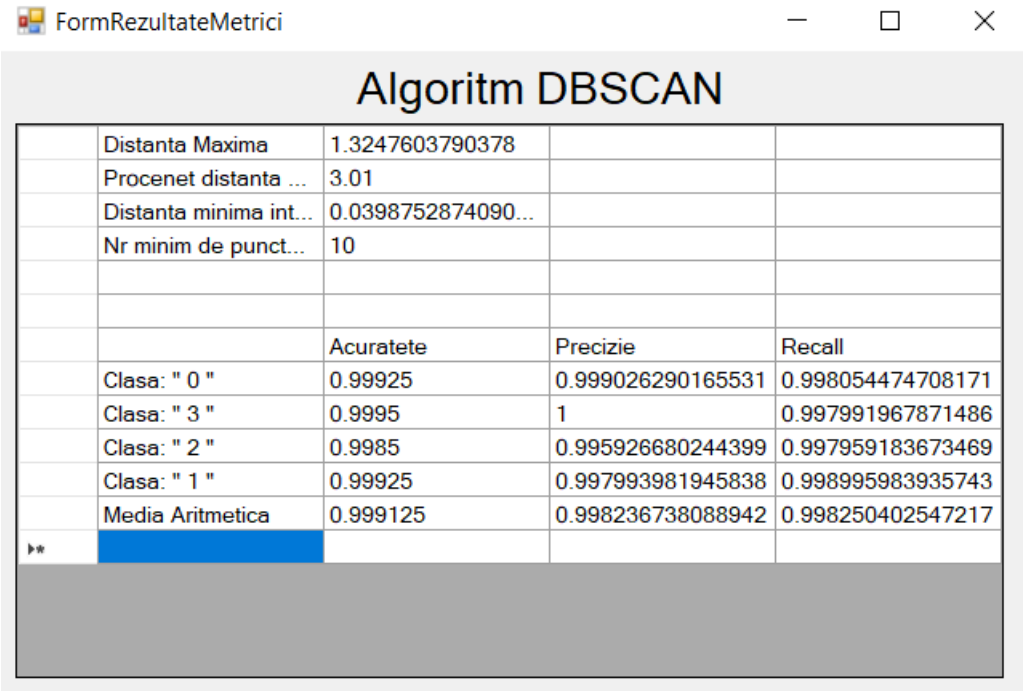


Fig. 3.40 Afișarea matricei de corespondență

La apăsarea butonului *Evaluare* se face matricea de eroare și se calculează cele trei metrici pe baza acesteia. La finalul acesteia se afișează o fereastră care conține parametri folosiți de algoritmi și metricile rezultate.



	Distanta Maxima	1.3247603790378		
	Procenet distanta ...	3.01		
	Distanta minima int...	0.0398752874090...		
	Nr minim de punct...	10		
		Acuratete	Precizie	Recall
	Clasa: " 0 "	0.99925	0.999026290165531	0.998054474708171
	Clasa: " 3 "	0.9995	1	0.997991967871486
	Clasa: " 2 "	0.9985	0.995926680244399	0.997959183673469
	Clasa: " 1 "	0.99925	0.997993981945838	0.998995983935743
	Media Aritmetica	0.999125	0.998236738088942	0.998250402547217
▶*				

Fig. 3.41 Afișarea metricilor

Butonul *Generare Excel* creează un fișier excel care conține informațiile din fereastra prezentată anterior.

4 Concluzii

După aplicarea procesului de mai multe ori, folosind diferite metode de similaritate, diferite metode de normalizare a datelor și diferite fișiere am tras mai multe concluzii despre cei doi algoritmi, despre datele cu care lucrează aceștia și cât de sensibili sunt la parametrii de intrare.

O primă concluzie este aceea că algoritmul HAC este mai eficient în clusterizarea punctelor, deoarece algoritmul DBSCAN a avut rezultate mai bune la recall și acurateț doar când s-a folosit distanța Minkowski și normalizarea Suma1. Algoritmul HAC nu este la fel de sensibil la parametrii de intrare ca algoritmul DBSCAN, dar algoritmul DBSCAN poate detecta zgomotul din date.

O a doua concluzie este aceea că normalizarea binară și normalizarea Cornell-Smart nu sunt utilizabile pentru fișierele care conțin puncte pe grafic.

O altă concluzie este aceea că distanța cosinus restrânge foarte mult domeniu de reprezentare a distanței, astfel încât distanța minimă între două puncte de la algoritmul DBSCAN sau pragul de la algoritmul HAC este foarte mic. La puncte este un procent de 0,035% din distanța maximă calculată.

Încă o concluzie este aceea că normalizarea Suma1 și Binară sunt greu utilizabile când se lucrează cu fișiere care conțin documente sub formă de vectori. Suma1 îngustează foarte mult domeniul de reprezentare a acestor date [0, 0.49]. Când se folosește normalizarea nominală și Min-Max, noul domeniu fiind [0,1], pe aceste fișiere intervalul rezultat este același.

O altă concluzie este aceea că datele din fișierele care conțin documente sub formă de vectori de frecvență de cuvinte nu sunt liniar separabile.Se poate observa că cei doi algoritmi nu au rezultate bune atunci când se utilizează acest tip de fișiere, algoritmul HAC având valori de 0 atât la precizie cât și la recall. Se poate observa și faptul că dacă este o valoare bună a acurateței este posibil să avem rezultate slabe ale preciziei și ale recall-ului.

Ca o concluzie generală pot spune că cei doi algoritmi nu au rezultate bune pe fișiere care conțin documente. Algoritmii au rezultate bune când utilizează fișiere care conțin puncte și că algoritmul HAC are rezultate mai bune decât algoritmul DBSCAN. Pentru clusterizarea punctelor cea mai bună metodă de normalizare este normalizarea Min-Max, iar cea mai bună metodă de calcul a distanței este distanța euclidiană. Aceste metode sunt cele mai bune pentru ambii algoritmi.

5 Bibliografie

[1] ACU D., ACU M., DICU P., ACU A., -Analiză matematică,Editura „ALMA MATER”, Sibiu, 2002

[2] AGGARWAL C., REDDY C., - Data clustering : algorithms and applications, Editura Taylor & Francis Group, 2014

[3] CREŢULESCU R., -Contribuţii la proiectarea sistemelor de clasificare a documentelor, Sibiu, 2011

[4] CREŢULESCU R., MORARIU D., - Text mining : tehnici de clasificare și clustering al documentelor, Editura Albastă, Cluj-Napoca, 2012

[5] HAN J., KAMBER M., -Data Mining: Concepts and Techniques Second Edition, Editura Elsevier Inc., San Francisco, 2006

[6] MOHRI M., ROSTAMIZADEH A., TALWALKER A., - Foundations of Machine Learning, second edition, Editura Massachusetts Institute of Technology, 2018

[7] MORARIU D., -Contribution to Automatic Knowledge Extraction from Unstructured Data, Sibiu, 2007

[8] https://en.wikipedia.org/wiki/Distance
(accesat în 30.05.2021)

[9] https://en.wikipedia.org/wiki/Minkowski_distance
(accesat în 30.05.2021)

[10] https://ro.wikipedia.org/wiki/Norm%C4%83_(matematic%C4%83)
(accesat în 30.05.2021)

[11] https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826
(accesat în 04.06.2021)

[12] https://www.slideshare.net/FlorinLeon/algoritmi-de-grupare-clustering
(accesat în 02.06.2021)