**Rezolvarea problemei de optimizare a unei masini cu vectori suport cu ajutorul unui algoritm evolutiv**

1. **Descrierea problemei considerate**

Problema optimizării unei masini cu vectori suport (SVM) este o problema fundamentala in invatarea automata, utilizata pentru clasificarea datelor. SVM este un algoritm de învatare supravegheata care cauta sa gaseasca un hyperplan optim pentru a separa două clase de date. In aceasta problema, scopul este de a optimiza parametrii unui model SVM utilizand un algoritm evolutiv.

Algoritmul evolutiv reprezinta o abordare inspirationala din natura evolutiei biologice, utilizand operatii precum selectia, incrucisarea si mutatia pentru a explora un spatiu de solutii. In cadrul acestui proiect, algoritmul evolutiv este folosit pentru a ajusta coeficientii vectorilor de sprijin (alpha) intr-un SVM, cu scopul de a maximiza performanta modelului.

Solutia propusa va utiliza un set de date (cum ar fi Wine Quality Dataset) pentru a antrena un model SVM si va aplica un algoritm evolutiv pentru a optimiza parametrii sai.

Disclaimer: In problema noastra, dimensiunea populatiei am luat-o in jur de 30. Deoarece pentru mai mult de 100, calculatorul ramane fara memorie si se blocheaza.

1. **Aspecte teoretice privind algoritmul**

Un SVM (Support Vector Machine) este un algoritm care cauta sa gaseasca un hyperplan optim ce separa doua clase de date. SVM cauta acei vectori din setul de date care sunt cele mai apropiate de hyperplanul de separare, acestia fiind numiti vectori de sprijin (support vectors).

In cadrul aceste probleme, optimizarea SVM presupune ajustarea coeficientului alpha pentru fiecare vector de sprijin, iar scopul este de a minimiza o functie de cost. Algoritmul evolutiv va fi folosit pentru a gasi cea mai buna solutie pentru acesti coeficienti alpha, utilizand urmatoarele concepte:

**1. Initializarea populatiei:** Valorile initiale sunt cele date de pe UCL Repository (Wine Quality)

**2. Selectia:** Selectia parintilor se face pe baza unui criteriu de fitness (in cazul nostru, performanta functiei de cost)

**3. Crossover (Incrucisarea):** Doua solutii parentale sunt combinate pentru a crea o noua solutie, oferind astfel diversitate in populatia solutiilor.

**4. Mutatia:** Pentru a asigura diversitatea populatiei, se aplica transformari cu caracter aleatoriu asupra copiilor nou generati, permitand aparitia unor trasaturi care nu ar fi aparut in cadrul populatiei doar prin selectie si incrucisare.

Un pas esential in implementarea SVM este ajustarea valorilor alpha astfel incat sa respecte constrangerile impuse de SVM, care sunt intre 0 si o valoare maxim C. Algoritmul evolutiv va ajusta acesti parametrii astfel incat functia de cost sa fie minimizata, iar separarea claselor sa fie optimizata.

**Tipurile de codificari, selectii, incrucisare si mutatie utilizate:**

**Codificare:**

In algoritmii evolutivi, codificarea reprezinta modul in care solutiile candidate (indivizii) sunt reprezentate pentru a fi procesate de operatori de selectie, incrucisare si mutare. In program, codificarea care s-a folosit este cea reala.

* **Descriere:** Fiecare individ este reprezentat ca un vector de valori reale, corespunzatoare coeficientilor α pentru fiecare exemplu din setul de date: **ex:**
* **Avantaje:** 
  + Este intuitiva pentru probleme cu variabile continue (precum coeficientii α din problema SVM).
  + Nu necesita conversii intre reprezentari.
  + Permite explorarea mai eficienta a spatiului solutiilor reale.
* **Dezavantaje:**
  + Operatorii trebuie adaptati pentru variabile reale (de exemplu, mutatia si incrucisarea).
  + Poate necesita mecanisme pentru a respecta constrangerile (cum ar fi 0 ≤ α ≤ C)

**Selectie**

In cod, selectia parintilor a fost realizata printr-o selectie bazata pe competitie (tournament selection). Aceasta presupune alegerea a doi indivizi aleatori din populatie si selectarea celui cu fitness mai bun pentru a deveni parinte.

**Avantaje:**

* Este simplu de implementat.
* Mentine presiunea de selectie moderata, permitand indivizilor cu fitness inferior sa aiba sansa sa contribuie, ceea ce ajuta la diversitate genetica.

**Dezavantaje:**

* In functie de dimensiunea competitiei (de obicei 2 sau mai multi indivizi), poate introduce sau reduce presiunea evolutiva, ceea ce poate afecta viteza de convergenta.

**Incrucisare**

Codul utilizeaza o forma de incrucisare reala cu un punct de crossover. In aceasta metoda:

1. Un punct de taieri este ales alatoriu de-a lungul vectorilor parintilor.
2. Partea de dinaintea punctului de taiere dintr-un parinte este combinata cu partea de dupa punctul de taiere din celalalt parinte pentru a crea un nou individ (si invers pentru al doilea copil).

**Avantaje:**

* Conserva structura solutiilor parintilor.
* Este simplu si rapid de calculat.

**Dezavantaje:**

* Este posibil sa nu exploreze spatiul solutiilor suficient de divers in comparatie cu alte metode de incrucisare (cum ar fi incrucisarea uniforma).

**Mutatie**

Codul implementeaza o **mutatie reala.** In aceasta metoda:

1. Se alege un indice aleatoriu al vectorului unui individ
2. Valoarea de la acel indice este modificata prin adaugarea unei valori aleatorii dintr-un interval predefinit (-0.5 pana la +0.5 in cazul dat).
3. Noua valoare este “taiata” pentru a respecta constrangerile (0 ≤ α ≤ C).

**Avantaje:**

* Introduce variatie continua, permitand explorarea fina a spatiului solutiilor.
* Este potrivita pentru probleme cu variabile reale, cum sunt coeficientii α din problema SVM.

**Dezavantaje:**

* Daca rata de mutatie este prea mica, explorarea devine limitata.
* Daca rata de mutatie este prea mare, poate destabiliza populatia, ducand la pierderea convergentei.

**Modalitatea de rezolvare**

Pentru a rezolva aceasta problema, s-au folosit urmatoarele etape:

1. **Preprocesarea datelor:**

* Setul de date Wine Quality a fost utilizat pentru antrenarea si testarea modelului SVM. Etichetele calitatii vinului au fost transformate intr-un set de date binar (calitate > 7 este etichetata ca -1, iar restul ca 1)
* Datele au fost scalate folosind StandardScaler pentru a le normaliza.

1. **Reducerea dimensiunii:**
   * Dimensiunea datelor a fost redusa folosind PCA (Principal Component Analysis), pentru a imbunatati performanta algoritmului si a reduce complexitatea.
2. **Algoritmul evolutiv:**
   * Populatia de solutii a fost initializata aleatoriu. Fiecare individ reprezinta un vector alpha, care contine coeficientii pentru fiecare vector de sprijin.
   * Operatiile de selectie, crossover si mutatie au fost aplicate pentru a obtine o noua generatie de solutii.
   * Fitness-ul fiecarei solutii a fost evaluat pre baza functiei de cost SVM, iar cele mai bune solutii au fost selectate pentru a produce noi indivizi.
3. **Evaluarea modelului:**
   * Modelul a fost antrenat si evaluat folosind vectorii de sprijin gasiti in urma optimizarii evolutive, iar numarul de vectori de sprijin a fost calculat pentru a analiza eficienta modelului.

**4. Listarea partilor semnificative din codul sursa insotite de explicatii si comentarii**

def initialize\_population(size, n\_samples):  
 *"""  
 Functia care inițializeaza populatia  
 """* return [np.random.uniform(0, C, n\_samples) for \_ in range(size)]  
  
  
def crossover(parent1, parent2):  
 *"""  
 Functia de crossover care combină doi părinti pentru a crea copii  
 """* point = np.random.randint(0, len(parent1))  
 child1 = np.concatenate((parent1[:point], parent2[point:]))  
 child2 = np.concatenate((parent2[:point], parent1[point:]))  
 return child1, child2  
  
  
def mutate(individual):  
 *"""  
 Functia care efectuează mutatia asupra unui individ  
 """* index = np.random.randint(0, len(individual))  
 individual[index] += np.random.uniform(-0.5, 0.5)  
 individual[index] = np.clip(individual[index], 0, C)  
 return individual  
  
  
def select\_parent(population, fitness):  
 *"""Functia de selectie a parintilor pe baza fitness-ului"""* idx1, idx2 = np.random.choice(len(population), 2, replace=False)  
 return population[idx1] if fitness[idx1] > fitness[idx2] else populat

Aceste functii sunt esentiale pentru implementarea algorimului evolutiv:

* initialize\_population() creeaza solutiile initiale aleatorii. Populatia este formata dintr-un numar specificat de indivizi fiecare fiind un vector de dimensiune , unde este numarul total de esantioane din setul de date. Valorile coeficientilor α sunt initializate aleatoriu in intervalul [0, C], unde C este un hiperparametru care controleaza flexibilitatea modelului SVM.
* select\_parent() selecteaza parintii pentru crossover pe baza fitness-ului. Se aplica metoda competitiei, in care doi indivizi sunt selectati aleatoriu din populatie, iar cel cu fitness mai bun este ales ca parinte.
* crossover() creeaza noi solutii prin combinarea a doua solutii parentale. Se alege un punct de taiere aleatoriu in vectorul parintilor. Copilul 1 este creat prin combinarea primei portiuni din parintele 1 si a celei de-a doua portiuni din parintele 2. Copilul 2 este creat prin combinarea primei portiuni din parintele 2 si a celei de-a doua portiuni din parintele 1.
* mutate() introduce o schimbare aleatorie intr-o solutie. Se selecteaza un indice aleatoriu din vectorul individului. Valoarea la acel indice este ajustata cu o valoare aleatorie dintr-un interva presetat ([-0.5, 0.5] in cazul de fata). Dupa mutatie, valoarea este “taiata” pentru a respecta constrangerile [0, C].

**5. Rezultate obtinute prin rularea programului in diverse situatii, capturi de ecran si comentarii asupra rezultatelor obtinute**

Rezultatele obtinute in urma rularii programului au fost masurate prin evolutia valorii fitness-ului in fiecare generatie. Intr-o serie de 5 generatii, fitness-ul a avut valori variabile, cu o imbunatatire a acestuia pe parcursul evolutiei.

A screenshot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a number

Description automatically generated

In urma aplicarii algoritmului evolutiv, s-a observat o imbunatatire a performantei modelului, iar numarul de vectori de sprijin a fost redus comparativ cu solutiile initiale:

**Reducerea numarului de vectori de sprijin:**

* Optimizarea evolutiva a permis selectarea unui subset mai mic de vectori de sprijin, ceea ce reduce complexitatea modelului.
* Un model SVM cu mai putini vectori de sprijin este mai eficient in termeni de timp de predictie, fara a compromite semnificativ acuratetea.

**Imbunatatirea fitness-ului:**

* Valorile fitneess-ului solutiilor s-au imbunatatit treptat de-a lungul generatiilor, indicand o convergenta catre solutii mai bune.
* Aceasta evolutie demnostreaza ca operatorii de selectie, incrucisare si mutatie au fost eficienti in explorarea spatiului solutiilor.

**Stabilitatea modelului:**

* Dupa mai multe generatii, s-a observat o stabilizare a valorilor fitness, ceea ce sugereaza ca algoritmul a ajuns intr-un punct optim sau aproape optim.
* Aceasta stabilitate reflecta echilibrul dintre explorarea si exploatarea solutiilor, un aspect esential al algoritmilor evolutivi.

**Observatii aditionale:**

* Reducerea numarului de vectori de sprijin implica si o generalizare mai buna a modelului, minimizand riscul de overfitting.
* Algoritmul a demonstrat ca poate fi adaptat pentru alte probleme similare de optimizare, ceea ce ii creste versatilitatea.

**6. Concluzii**

* Algoritmul evolutiv s-a dovedit a fi o metoda eficienta pentru optimizarea unui model SVM, avand in vedere ca a reusit sa imbunatateasca performanta modelului prin ajustarea parametrilor alpha.
* Eficienta si complexitatea algoritmului pot fi influentate de parametrii de evolutie, cum ar fi dimensiunea populatiei, rata de mutatie si crossover.
* Pentru imbunatatiri viitoare, ar fi util sa se exploreze diverse functii de selectie si sa se testeze alte metode de optimizare a parametrilor SVM.

**Posibile imbunatatiri:**

1. **Selectie bazata pe ruleta (roulette wheel selection):** Poate fi utilizata pentru a acorda sanse proportionale cu fitness-ul, asigurand o diversitate mai mare a parintilor.
2. **Incrucisare uniforma:** In loc de un singur punct, se alege aleatoriu pentru fiecare gena daca provne de la parintele 1 sau parintele 2, ceea ce poate creste diversitatea solutiilor.
3. **Mutatie adaptiva:** Ajustarea dinamica a magnitudinii mutatiei in functie de progresul generatiilor (mutare mai mare la inceput si mai mica pe masura ce algoritmul converge).
4. **Normalizarea valorilor α:** Inainte de aplicarea algoritmului, se poate normaliza intreaga populatie pentru a asigura ca solutiile respecta constrangerile.
5. **Reprezentari hibride:** Daca problema devine mai complexa, se poate combina codificarea reala cu alte forme (ex: binara pentru anumite categorii discrte de parametrii).

**7. Bibliografie**

1. **Cortes, C., & Vapnik, V. (1995).** Support-vector networks. Machine learning, 20(3), 273-297.

2. **Goldberg, D. E. (1989).** Genetic algorithms in search, optimization, and machine learning.

3. **Shalev-Shwartz, S., & Ben-David, S. (2014).** Understanding machine learning: From theory to algorithms. Cambridge University Press.

**8. Lista cu ce a lucrat fiecare membru al echipei**

* **Nicodei Radu-George:** tot proiectul.
* **Bahrim Casian:** a inghetat anul