

Proiect la Inteligență artificială

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

Proiect realizat de:
Vrabie Vasile
Galan Ionuț
Arteni Elena-Denisia
Grupa: 1410A
Anul: IV
Specializare: TI

Descrierea problemei considerate

Problema considerată este optimizarea unei mașini cu vector suport utilizând un algoritm evolutiv. Mașinile cu vector suport reprezintă o metodă de învățare cu performanțe foarte bune pentru probleme de clasificare. La baza acestei metode stă conceptul de maximizare a marginii de separare a două clase. Pentru determinarea marginii se va rezolva problema duală utilizând un algoritm evolutiv.

Ca aplicație pentru această problemă am ales detectarea gradului de ocupare al unei camere, folosind un set de date. Setul conține un număr de date ce pot fi utilizate pentru antrenare și două fișiere cu date ce pot fi utilizate pentru testarea performanței aplicației. O instanță este reprezentată de șase atribute ce exprimă rezultate reale ale unor măsurători și ieșirea sau clasa corespunzătoare acelor valori. Astfel folosind data, temperatura, umiditatea, luminozitatea, cantitatea de dioxid de carbon și rația de umiditate se va putea determina dacă acea cameră considerată este ocupată sau nu.

Aspecte teoretice privind algoritmul

Mașinile cu vector suport urmăresc determinarea unei margini de separare a două clase. Existând mai multe limite de separație posibile, se pune problema care dintre acestea este mai bună. Una din ideile de bază la acest algoritm este că un model care generalizează cel mai bine, adică este capabil să clasifice instanțe noi în clasele lor corespunzătoare, este cel care desparte cel mai mult clasele. Astfel limita de decizie este cât mai departe de ambele clase. Marginea este distanța dintre dreptele paralele la acea dreaptă de decizie care ating cel puțin una din instanțele fiecărei clase. Deci se poate spune că algoritmul urmărește maximizarea marginii.

Considerăm formula:

$$h(\mathbf{x}) = g(\mathbf{w}^T \mathbf{x} + b) \quad g(z) = \begin{cases} 1, & \text{dacă } z \geq 0 \\ -1, & \text{dacă } z < 0 \end{cases}$$

Unde x este o instanță ce trebuie clasificată, w reprezintă un vector de ponderi corespunzătoare, b este "bias-ul", h este rezultatul funcției g care stabilește clasa din care face parte vectorul. Reprezentarea problemei pentru SVM presupune ca ecuațiile suprafețelor de separare să fie normalizate, astfel încât limitele ce trec prin vectorii suport ai celor două clase să fie $(w * x) + b = -1$ respectiv $(w * x) + b = 1$.

Rezolvarea problemei de clasificare se va reduce la determinarea parametrilor w și b astfel încât marginea $m = 2 / \|w\|$ să fie maximă pentru toate instanțele din mulțimea de antrenare. Această problemă de optimizare se numește problema primară. Echivalentă acestei rezolvări este rezolvarea problemei duale ce este în general mai ușor de rezolvat. Formularea acestei probleme se regăsește în imaginea de mai jos.

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle. \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \dots, m \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0, \end{aligned}$$

Prin rezolvarea problemei duale se vor determina multiplicatorii lagrangieni α , care vor fi 0 cu excepția celor corespunzători vectorilor suport ce vor avea valori reale pozitive. Pentru maximizarea acestei funcții și determinarea multiplicatorilor se folosește un algoritm evolutiv. Acesta este o metodă de căutare prin analogie cu selecția naturală biologică. Algoritmul are o populație de soluții posibile care evoluează iterativ prin aplicarea unor operatori. Evoluția se bazează pe baza presiunii evolutive, adică favorizarea indivizilor mai adaptați. Operatorii genetici sunt selecția, încrucișarea și mutația.

Un algoritm evolutiv are drept componente gena și cromozomul ce poate fi un vector de gene. O primă fază este inițializarea populației ce reprezintă o mulțime de cromozomi sau indivizi cu valori ale genelor aleatoare, dar în domeniul lor de definiție. Evaluare calității unei soluții sau exprimarea gradului de adaptare al unui individ se realizează printr-o funcție de "fitness" din care rezultă o valoare comparabilă cu altele. Următoarea fază este aplicarea operatorului de selecție ce va alege un părinte pe baza funcției de adaptare. Se poate utiliza elitismul ce constă în copierea celui mai adaptat individ direct în noua populație pentru a ne asigura că nu vom pierde cea mai bună soluție. După selecție are loc operatorul de încrucișare care combină doi părinți selectați și produce unul sau mai mulți cromozomi copil. Aceasta se realizează cu o probabilitate definită de utilizator. Pentru păstrarea diversității se aplică unui nou copil operatorul de mutație ce va modifica una sau mai multe gene după o probabilitate.

Aceste faze cu excepția celei de inițializare se repetă iterativ un număr de generații stabilit de utilizator sau până la atingerea unei precizii sau soluții dorite.

Astfel cu algoritmul evolutiv vom găsi valorile α și vectorii suport ce maximizează marginea și pe care îi putem folosi mai departe în calcule. În cazul problemelor care nu sunt separabile liniar va fi necesară transformarea datelor din spațiul inițial al problemei într-un spațiu cu mai multe dimensiuni unde clasele devin separabile liniar.

Separarea spațiului bidimensional în mai multe spații se realizează prin trucul nucleului. Vom folosi un nucleu gaussian care împarte spațiul valorilor date într-o infinitate de dimensiuni.

$$K_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \exp \left[-\gamma \|\mathbf{x} - \mathbf{x}'\|^2 \right]$$

Nucleul gaussian este cunoscut și sub denumirea de Radial basis function kernel sau RBF. Nucleul funcției calculează produsul între doi vectori proiectați.

$$K(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$$

Variabilele sunt γ (gamma), parametru setat de utilizator în care se stabilește răspândirea nucleului și ψ , funcție care proiectează vectorii \mathbf{x} într-un nou spațiu vectorial.

Următoarea etapă după separarea datelor din spațiul inițial al problemei folosind trucul nucleului este să calculăm limita de decizie în cazul claselor datelor de intrare. Distanța dintre cele 2 clase reprezintă marginea dintre doi vectori proiectată pe un vector nou \mathbf{w} . Determinarea valorilor se fac rezolvând problema duală care este în general mai ușor de rezolvat decât problema primară.

$$w = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)},$$

$$b = \frac{1}{|S|} \sum_{s \in S} \left(y_s - \sum_{t \in S} \alpha_t y_t (\mathbf{x}_t \cdot \mathbf{x}_s) \right)$$

Variabile utilizate sunt: \mathbf{b} , număr real echivalent unui prag și utilizat în calculul discriminantului, \mathbf{S} mulțimea vectorilor suport și $\mathbf{x}^t \cdot \mathbf{x}_s$, este funcția nucleului gaussian. Calcularea funcției discriminant pentru vectorul valorilor de intrare:

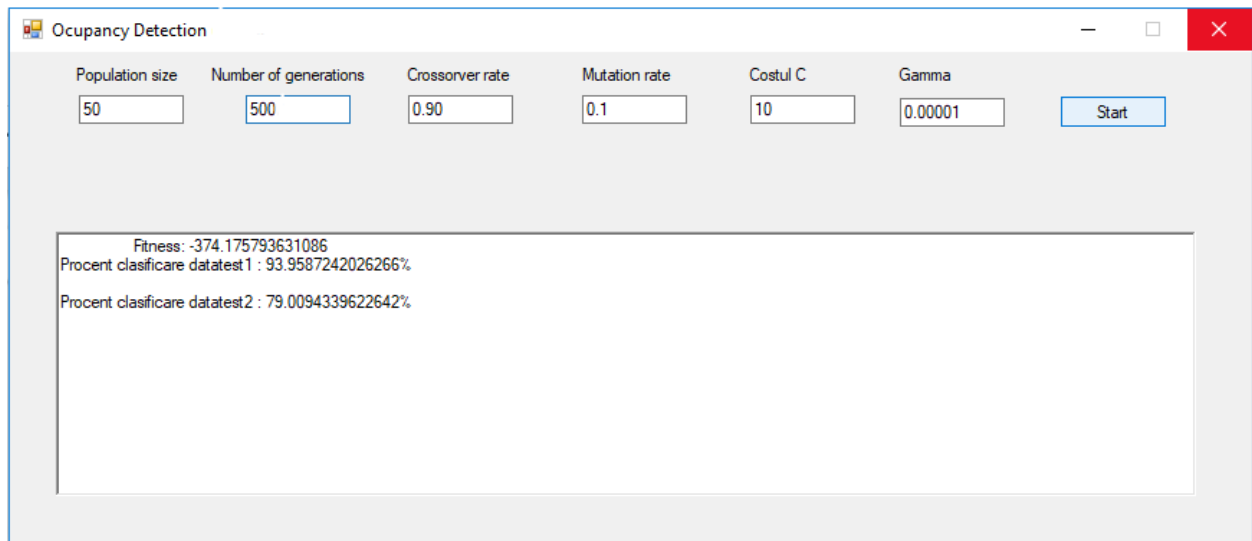
$$f(\mathbf{x}) = \sum_{i=1}^m \alpha_i y^{(i)} K(\mathbf{x}^{(i)}, \mathbf{x}) + b$$

Variabile:

- \mathbf{m} dimensiunea vectorului de intrare
- α valoarea vectorilor suport
- \mathbf{y} clasa din care face parte vectorul corespondent
- $\mathbf{K}(\mathbf{x}^{(i)} + \mathbf{x})$ funcția nucleului gaussian
- \mathbf{b} pragul

Modalități de rezolvare

Aplicația este de tip Windows Forms în limbajul de programare C#. La pornirea acesteia va apărea interfața cu utilizatorul de unde vor putea fi setați parametrii și unde se va vizualiza rezultatul.



La apăsarea butonului Start se vor citi parametrii și se va executa algoritmul. În primul rând se vor citi datele de antrenare, se vor transforma în valori reale cele ce nu sunt (data), și clasele vor fi transformate din 0 în -1, din 1 va rămâne 1. Următorul pas este apelarea algoritmului evolutiv și salvarea celei mai bune soluții.

Clasa Chromosome are structura:

```
public class Chromosome
{
    public int noGenes { get; set; } // egal cu numarul de instante de antrenare

    public double[] alfa { get; set; } // valorile alfa i sau genele

    public double minValue { get; set; } //valoarea minima pe care e definit alfa
    public double maxValue { get; set; } //valoarea maxima adica parametrul C

    public double Fitness { get; set; } // valoarea functiei de adaptare a individului

    private static Random _rand = new Random();
}
```

Tipul selecției folosite este cea prin competiție unde se aleg doi indivizi diferiți din populație și se selectează cel mai adaptat dintre ei. Se folosește și elitismul ce constă în copierea celei mai bune soluții direct în noua populație.

```
public static Chromosome Tournament(Chromosome[] population)
{
    int indexi = _rand.Next(0, population.Length);
    int indexj;
    do
    {
        indexj = _rand.Next(0, population.Length);
```

```

} while(indexi == indexj);      //cat timp indivizii sunt diferiti
if (population[indexi].Fitness > population[indexj].Fitness)
    return new Chromosome( population[indexi]);

return new Chromosome(population[indexj]);
}

```

Pentru încrucișarea reală am folosit încrucișarea aritmetică unde se creează copii între părinți după formula: $z_i = x_i + (1 - \alpha) y_i$, cu $0 \leq \alpha \leq 1$.

```

public static Chromosome Arithmetic(Chromosome mother, Chromosome father, double rate)
{
    double prop = _rand.NextDouble();
    Chromosome child;
    if (prop < 0.5) //daca nu se face incrucisarea atunci se alege unul din parinti cu
    probabilitatea de 50%
        child = new Chromosome(mother);
    else
        child = new Chromosome(father);

    double probability = _rand.NextDouble();
    if (probability <= rate) //daca se face incrucisarea se vor calcula valorile cu multiplicatorul
    aleatoriu intre 0,1
    {
        double multiplicator = _rand.NextDouble();

        for(int j =0; j<mother.noGenes;j++)
        {
            child.alfa[j] = multiplicator * mother.alfa[j] + (1 - multiplicator) * father.alfa[j];
        }
    }

    return child;
}

```

Operatorul de mutație va reseta o genă în domeniul ei de definiție cu o probabilitate:

```

public static void Reset(Chromosome child, double rate)
{
    for (int i = 0; i < child.noGenes; i++)
    {
        double prob = _rand.NextDouble();
        if (prob < rate)
        {
            child.alfa[i] = child.minValue + _rand.NextDouble() * (child.maxValue -
            child.minValue);
        }
    }
}

```

Funcția ce calculează fitness-ul este modelarea formei duale ce se dorește a fi maximizată:

```

public static void ComputeFitness(Chromosome chromosome, DataSet dataSet, double gamma)
{
    //trebuie fortata constrangerea, ajustam valorile afla ca suma de alfai * yi = 0 ;
    double []newAlfa = utils.Util.AdjustmentAlgorithm(chromosome.alfa, dataSet.y);
}

```

```

chromosome.alfa = newAlfa;

double suma1 = 0.0; //prima suma din functie
double suma2 = 0.0; //adoua suma din functie

for (int i = 0; i < newAlfa.Length; i++)
{
    suma1 += newAlfa[i];
    for (int j = 0; j < newAlfa.Length; j++)
    {
        if (newAlfa[j] != 0.0 && newAlfa[i] != 0.0)
        {
            suma2 += dataSet.y[i] * dataSet.y[j] * newAlfa[i] * newAlfa[j] *
utils.Util.gaussianKernel(dataSet.instanta[i].x, dataSet.instanta[j].x, gamma);
        }
    }
}

double fitness = suma1 - 0.5 * suma2; //forma duala
chromosome.Fitness = fitness;
}

```

Un aspect important este respectarea constrângerii problemei duale ce spune că suma $\alpha_i y_i = 0$, $0 \leq i \leq \text{numărul de instanțe}$. Astfel în algoritmul de ajustare se calculează suma coeficienților lagrangieni pozitivi și suma celor negativi. Dacă suma pozitivă este mai mare decât suma negativă înseamnă că trebuie să reducem acea sumă micșorând multiplicatorii corespunzători. Pentru acest lucru alegem aleatoriu un multiplicator pozitiv, iar dacă acesta este mai mic decât suma noastră îl facem 0, iar dacă este mai mare înseamnă că trebuie să reducem acel multiplicator cu diferența dintre suma pozitivă și cea negativă pentru ca acestea să devină egale și să satisfacă constrângerea, și invers în caz contrar, până este satisfăcut criteriul de oprire.

```

public static double[] AdjustmentAlgorithm(double[] oldAlfa, double[] y)
{
    double[] newAlfa = new double[oldAlfa.Length];

    for (int i = 0; i < oldAlfa.Length; i++)
    {
        newAlfa[i] = oldAlfa[i]; //copiez vechile valori
    }
    double suma = 0.0;
    double sumaPozitiva = 0.0;
    double sumaNegativa = 0.0;
    for (int j = 0; j < newAlfa.Length; j++)
    {
        if (newAlfa[j] != 0.0)
        {
            if (y[j] == 1)
                sumaPozitiva += newAlfa[j];
            else
                sumaNegativa += newAlfa[j];
        }
    }
    suma = sumaPozitiva - sumaNegativa;
    while (suma != 0.0) //criteriul de stop

```

```

{
    if (suma < 0) suma = -suma; //conteaza valoarea, nu semnul
    int indexk;
    if (sumaPozitiva > sumaNegativa) //unde suma este mai mare trebuie sa gasesc coeficienti
    si sa ii scad
    {
        do
        {
            indexk = _rand.Next(0, newAlfa.Length); //aleg random un index

            } while ((y[indexk] != 1.0) || (newAlfa[indexk] == 0.0)); //aleg o valoare cu plus care
    are un alfa pozitiv
            if (newAlfa[indexk] > suma) //atunci il reduc cu suma si astfel sumaPozitiva va deveni
    egala cu sumaNegativa si se indeplineste criteriul
            {
                newAlfa[indexk] = newAlfa[indexk] - suma; //scad suma din coeficient
                sumaPozitiva -= suma; //actualizez suma coeficientilor pozitivi care scade cu cat a
    scazut si coeficientul
            }
            else
            {
                sumaPozitiva -= newAlfa[indexk]; //din suma pozitiva se scade cat era in newalfa la
    pozitia indexk
                newAlfa[indexk] = 0.0; //micsorez suma facand coeficientul 0
            }
        }
        else
        {
            do
            {
                indexk = _rand.Next(0, newAlfa.Length);

                } while ((y[indexk] != -1.0) || (newAlfa[indexk] == 0.0)); //aleg o valoare cu minus
            if (newAlfa[indexk] > suma)
            {
                newAlfa[indexk] = newAlfa[indexk] - suma; //scad suma din coeficient
                sumaNegativa -= suma; //actualizez suma coeficientilor pozitivi
            }
            else
            {
                sumaNegativa -= newAlfa[indexk]; //din suma pozitiva se scade cat era in newalfa la
    positia indexk
                newAlfa[indexk] = 0.0;
            }
        }
    }

    suma = sumaPozitiva - sumaNegativa;

}

return newAlfa;
}

// calcularea functiei kernel gaussian

```


// parametrii sunt valoarea vectorilor de intrare iar gamma este un parametru ales

```
public static double gaussianKernel(double[] x, double[] z, double gamma)
{
    double sum = 0.0;
    for (int i = 0; i < x.Length; ++i)
        sum += (x[i] - z[i]) * (x[i] - z[i]);

    return Math.Exp(-gamma * sum);
}
```

*// determinare valoare reala **b** care este o valoare de prag*

```
public static double ComputeB(double[] alfa, InputData[] x, double[] y, double gamma)
{
    int S = 0; //numarul vectorilor suport
    double sumat = 0.0, sumas = 0.0;
    for (int s = 0; s < alfa.Length; s++)
    {
        if (alfa[s] != 0.0) //atunci este vector suport
        {
            S++;
            sumat = 0.0;
            for (int t = 0; t < alfa.Length; t++)
            {
                if (alfa[t] != 0.0)
                {
                    sumat += alfa[t] * y[t] * Utils.Util.gaussianKernel(x[t].x, x[s].x, gamma);
                }
            }
            sumas += y[s] - sumat;
        }
    }
    return (1.0 / (double)S) * sumas;
}

//functia Discriminant este folosita in clasificare
public static double functieDiscriminant(double[] x, double[] alfa, DataSet dataset, double b,
double gama)
{
    double suma = 0.0;
    for (int i = 0; i < alfa.Length; i++)
    {
        if (alfa[i] != 0.0) //altfel produsul ar fi 0
        {
            // x sunt valorile ce trebuie clasificate
            suma += alfa[i] * dataset.y[i] * Utils.Util.gaussianKernel(dataset.instanta[i].x, x,
gama);
        }
    }
    return suma + b;
}
```

Testare

Pentru a evidenția eficiența rezultatului se calculează raportul (procentul de clasificare) dintre rezultatele obținute în urma aplicării funcției discriminant prezentată mai sus și rezultatele așteptate. La fiecare testare se utilizează în jur de 2000-9000 de instanțe pentru testare, în urma cărora rezultă două rapoarte, primul fiind calculat în urma procesării a 2000 de instanțe, iar al doilea utilizează 9000 de instanțe.

Ne-am propus să observăm cum se va modifica rezultatele în urma modificării **costului** între valorile {10, 25, 40, 55, 70, 85, 100} și a parametrului **gama** între valorile {0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0, 1.1}. Pentru a efectua testarea dorită am setat **populația inițială: 50, numărul de generații : 7000, rata de încrucișare: 0.9 și posibilitatea de mutație: 0.1**, în urma unor teste, aceste date de intrare au arătat cel mai mare procent de clasificare.

The screenshot shows the 'Occupancy Detection' application window. The parameters are set as follows:

Population size	Number of generations	Crossover rate	Mutation rate	Costul C	Gamma
35	6000	0.90	0.1	10	1E-05

The results displayed are:

```
Fitness: -6.98446415483756
Procent clasificare datatest1 : 93.8836772983114%
Procent clasificare datatest2 : 95.4778506972929%
```

Testat cu populatia: 35 și numărul de generații: 6000.

The screenshot shows the 'Occupancy Detection' application window with the following parameters:

Population size	Number of generations	Crossover rate	Mutation rate	Costul C	Gamma
50	7000	0.90	0.1	10	1E-05

The results displayed are:

```
Fitness: -52.4912041525579
Procent clasificare datatest1 : 96.3602251407129%
Procent clasificare datatest2 : 98.3695652173913%
```

Testat cu populatia: 70 și numărul de generații: 7000.

Se poate observa că în această situație procentajul de calculare corectă a rezultatului este mai mare. După testarea cu modificarea parametrului gama am obținut următoarele rezultate:

Test: 1

{

Input:

populationSize: 50
maxGenerations: 7000
crossoverRate: 0.90
mutationRate: 0.1
cost: 10
gama: 1E-05

Output:

Fitness: -99.9495808622225
Procent clasificare datatest1: 94.8217636022514
Procent clasificare datatest2 : 94.4114027891714

Test: 2

{

Input:

populationSize: 50
maxGenerations: 7000
crossoverRate: 0.90
mutationRate: 0.1
cost: 10
gama: 0.0001

Output:

Fitness: -71.1777062635293
Procent clasificare datatest1: 90.9193245778612
Procent clasificare datatest2 : 91.304347826087

}

Test: 3

{

Input:

populationSize: 50
maxGenerations: 7000
crossoverRate: 0.90
mutationRate: 0.1
cost: 10
gama: 0.001

Output:

Fitness: -39.2477520462377
Procent clasificare datatest1: 82.8893058161351
Procent clasificare datatest2 : 54.6554552912223

}

Test: 4

{

Input:

populationSize: 50
maxGenerations: 7000
crossoverRate: 0.90
mutationRate: 0.1

cost: 10
gama: 0.01

Output:

Fitness: -64.5865722420039
Procent clasificare datatest1: 75.046904315197
Procent clasificare datatest2 : 25.2153404429861

}

Test: 5

{

Input:

populationSize: 50
maxGenerations: 7000
crossoverRate: 0.90
mutationRate: 0.1
cost: 10
gama: 0.1

Output:

Fitness: -30.3112179702882
Procent clasificare datatest1: 74.8968105065666
Procent clasificare datatest2 : 24.2924528301887

}

Test: 6

{

Input:

populationSize: 50
maxGenerations: 7000
crossoverRate: 0.90
mutationRate: 0.1
cost: 10
gama: 1

Output:

Fitness: -4.51731021926156
Procent clasificare datatest1: 63.5272045028143
Procent clasificare datatest2 : 78.9889253486464

}

La testarea algoritmului cu parametrul gama neschimbat și modificarea parametrului cost a rezultat următoarele valori:

Test: 1

{

Input:

populationSize: 50
maxGenerations: 7000
crossoverRate: 0.90
mutationRate: 0.1
cost: 10
gama: 1E-05

Output:
Fitness: -99.9495808622225
Procent clasificare datatest1: 94.8217636022514
Procent clasificare datatest2 : 94.4114027891714
}

Test: 2
{
Input:
populationSize: 50
maxGenerations: 7000
crossoverRate: 0.90
mutationRate: 0.1
cost: 25
gama: 1E-05

Output:
Fitness: -274.671649430882
Procent clasificare datatest1: 96.6979362101313
Procent clasificare datatest2 : 99.3232157506153
}

Test: 3
{
Input:
populationSize: 50
maxGenerations: 7000
crossoverRate: 0.90
mutationRate: 0.1
cost: 40
gama: 1E-05

Output:
Fitness: -376.079338396296
Procent clasificare datatest1: 93.9587242026266
Procent clasificare datatest2 : 94.8010664479081
}

Concluzie

În urma testării putem observa că pentru a obține un procent cât mai mare de clasificare parametrul γ trebuie să fie < 0.0001 și populația de început > 45 , numărul de generații > 6000 . Se observă că parametrul C (costul) este o măsură a erorii admise. Creșterea valorii lui C mărește costul clasificării greșite a instanțelor și determină crearea unui model mai precis, dar care nu va generaliza bine. Pe de altă parte un parametru γ definește cât de mare este influența unui singur vector de antrenare, o valoare mică însemnând "departe", iar o valoare mare însemnând "aproape".

Astfel aplicația pune în evidență performanțele ridicate ale mașinilor cu vector suport pentru problemele de clasificare, simplitatea problemei duale și posibilitatea optimizării acestora utilizând algoritmi evolutivi ce sunt ușor de înțeles, găsesc optimul global și prezintă robustețe.

Bibliografie

Setul de date utilizat

<https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+?fbclid=IwAR0L40HXz7SHeHHMDhej56NoPIz9CyH7K6C5BVND6o4rGH30lkIGdAtuow8>.

Kernel trick

https://indico.cern.ch/event/439520/contributions/1941519/attachments/1147353/1645460/Kernel_trick_Deep_learning.pdf?fbclid=IwAR3QF510BArinsZUgl4q8uAgUSH8qiXh0M2aT2vfRZsOJPSzcMwbkiL-KXY

Florin Leon (2014). Inteligență artificială: mașini cu vectori suport, Tehnopress, Iași, ISBN 978-606-687-155-6, <http://florinleon.byethost24.com>

Inspirație pentru algoritmul de ajustare

https://www.researchgate.net/publication/309565420_Evolutionary_Support_Vector_Machines_A_Dual_Approach

Sarcini:

Arteni Elena-Denisia:

- Citire și interpretare a setului de date
- Realizare algoritm evolutiv

Galan Ionuț:

- Realizarea trucului kernel
- Calcularea parametrului b și a funcției discriminant

Vrabie Vasile:

- Realizarea clasificării și calcularea procentelor
- Testarea manuală și automată și interpretarea rezultatelor