INTELIGENȚĂ ARTIFICIALĂ

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

Clasificarea mașinilor

Autor: APETRIA George-Laurențiu

Cuprins

SECȚIUNEA 1: Descrierea problemei considerate

Problematica de bază a proiectului este reprezentată de clasificarea unui set de mașini care se împarte în 4 categorii: unacceptable, acceptable, good și very good, fiecare categorie având un anumit număr de instanțe în setul de antrenare, respectiv testare. Mai mult decât atât, fiecare instanță conține un număr de 6 atribute prin intermediul cărora se poate face clasificarea (buying, maint, doors, persons, lug_boot, safety). Soluția propusă pentru a rezolva problema menționată mai sus este utilizarea unui algoritm evolutiv cu ajutorul căruia să se determine parametrii SVM-ului, necesari ulterior în procesul de clasificare. După determinarea ecuației caracteristice SVM-ului (parametrii alpha, bias, W), ecuație care separă planul în două regiuni, poate începe procesul de clasificare.

SECȚIUNEA 2: Aspecte teoretice privind algoritmul

Din punct de vedere tehnic, trebuie rezolvată problema duală a SVM, care presupune o maximizare: W(alfa). Maximizarea se realizează cu ajutorul algoritmului evolutiv. SVM este la bază un perceptron cu un singur strat, deci va trebui să determinăm dreapta care maximizează marginea ce desparte cele două clase de obiecte. Datele (obiectele care sunt intersectate de margine) se numesc vectori suport. Sunt vectori deoarece sunt puncte într-un spațiu n dimensional – în acest caz 2 dimensiuni și sunt suport pentru că sprijină marginea. Scopul modelului este: detectarea valorilor suport care maximizează marginea. Deci există o singură margine de lățime maximă. Pentru algoritmul evolutiv, se utilizează varinata standard prezentată în curs.

Problema duală a SVM:

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

Genele din cromozomi vor fi alpha-urile, funcția de fitness va fi W(alfa).

Funcția de fitness:

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle.$$

Soluția va fi cromozomul care maximizează W, apoi din alpha-uri se refac parametrii **w** și b și apoi se face clasificarea.

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

Deoarece setul de date de antrenare are 1728 de instanțe – un număr foarte mare pentru a putea fi considerat ca număr de gene - fiecare individ α va fi compus din 100 de gene (numărul de gene este dat de numărul de instanțe, adică 100 (62 instanțe din clasa unacc, 20 instanțe din clasa acc, 9 instanțe din clasa good, 9 instanțe din clasa vgood)):

$$\alpha = [\alpha_1\ \alpha_2\ \alpha_3 \ldots\ \alpha_{100}]$$

În plus, fiecare individ trebuie să respecte două constrângeri:

$$0 \leq \alpha_i \leq C \ (1)$$

$$\sum_{i=1}^{n} \alpha_i\, y_i = 0 \ (2)$$

După ce obținem multiplicatorii Lagrange prin intermediul algoritmului genetic, se calculează bias-ul cu ajutorul fomulei:

$$b = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{m} \left(y_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j \right) \right)$$

Parametrii specifici SVM utilizați în cadrul implementării sunt:

- Nucleul polinomial de grad II.
- Costul C = 1e-5.

SECȚIUNEA 3: Modalitatea de rezolvare

Mașinile cu vector suport pleacă de la o fundamentare matematică riguroasă, deci nu sunt modele empirice în care se fac încercări pentru testarea funcționalității sau pentru modificarea numărului de straturi sau de neuroni pentru a mai face o încercare.

Ideea de bază a SVM este: cum putem avea garanții că un model obține soluții bune pe mulțimea de test?

Cheia problemei este că datele de test sunt din aceeași distribuție de probabilitate cu datele de antrenare. Acestea nu sunt complet diferite, ci trebuie să respecte o anumită structură. Așadar, atât datele de antrenare, cât și datele de test trebuie să fie eșantionate din aceeași distribuție de probabilitate, adică din aceeași clasă mare de obiecte.

SVM este prin excelență un clasificator binar, însă problema de clasificare ce trebuie rezolvată implică existența a patru clase distincte. Așadar, se impune o problemă de clasificare cu clase multiple. În acest sens, am optat pentru abordarea „una versus toate“. Astfel, am obținut cele 4 modele:

- unacc VS others
- acc VS others
- good VS others
- v-good VS others

Pentru fiecare dintre cele 4 modele se utilizează separat algoritmul evolutiv pentru a determina coeficienții Lagrange, respectiv bias-ul.

În final, pentru a clasifica o nouă instanță, se apelează toate cele 4 modele și se alege clasa cu funcția de decizie maximă:

$$C = \underset{i=1..k}{\operatorname{argmax}} \left(f(\mathbf{x}) \right)$$

SECȚIUNEA 4: Listarea părților semnificative din codul sursă însoțite de explicații și comentarii

// Cuantificarea unui set de date

```
String line;

// Class 1 set quantification - unacc
string quantified = "";
try
{
	// Pass the file path and file name to the StreamReader constructor
	StreamReader sr = new StreamReader("D:\\Support_Vector_Machine\\dataSet.data");
	// Read the first line of text
	line = sr.ReadLine();

	char[] delimiterChars = { ',' };
	int[] quantifiedValues = new int[7];
	string quantifiedValuesString = "";

	// Continue to read until you reach end of file
	while (line != null)
	{
		string[] words = line.Split(delimiterChars);

		switch (words[0])
		{
			case "low":
				quantifiedValues[0] = 1;
				break;
			case "med":
				quantifiedValues[0] = 2;
				break;
			case "high":
				quantifiedValues[0] = 3;
				break;
			case "vhigh":
				quantifiedValues[0] = 4;
				break;
		}

		switch (words[1])
		{
			case "low":
				quantifiedValues[1] = 1;
				break;
			case "med":
				quantifiedValues[1] = 2;
				break;
			case "high":
				quantifiedValues[1] = 3;
				break;
			case "vhigh":
				quantifiedValues[1] = 4;
				break;
		}
```

```
switch (words[2])
{
	case "2":
		quantifiedValues[2] = 1;
		break;
	case "3":
		quantifiedValues[2] = 2;
		break;
	case "4":
		quantifiedValues[2] = 3;
		break;
	case "5more":
		quantifiedValues[2] = 4;
		break;
}

switch (words[3])
{
	case "2":
		quantifiedValues[3] = 1;
		break;
	case "4":
		quantifiedValues[3] = 2;
		break;
	case "more":
		quantifiedValues[3] = 3;
		break;
}

switch (words[4])
{
	case "small":
		quantifiedValues[4] = 1;
		break;
	case "med":
		quantifiedValues[4] = 2;
		break;
	case "big":
		quantifiedValues[4] = 3;
		break;
}

switch (words[5])
{
	case "low":
		quantifiedValues[5] = 1;
		break;
	case "med":
		quantifiedValues[5] = 2;
		break;
	case "big":
		quantifiedValues[5] = 3;
		break;
}

if (words[6] == "unacc")
{
	quantifiedValues[6] = 1;
}
```

```
                else
                {
                        quantifiedValues[6] = -1;
                }

                for (int i = 0; i < 6; ++i)
                {
                        quantifiedValuesString += quantifiedValues[i].ToString();
                        quantifiedValuesString += ",";
                }
                quantifiedValuesString += quantifiedValues[6].ToString();

                quantified += quantifiedValuesString;
                quantified += '\n';

                // Read the next line
                line = sr.ReadLine();
                quantifiedValuesString = "";
        }

        using (StreamWriter sw = new
StreamWriter(@"D:\\Support_Vector_Machine\\dataSetQuantifiedUnacc.data"))
        {
                quantified = quantified.Trim();
                sw.Write(quantified);
        }

        // Close the file
        sr.Close();
}

catch (Exception e)
{
        Console.WriteLine("Exception: " + e.Message);
}
finally
{
        Console.WriteLine("Writing to the dataSetQuantifiedUnacc.data file was completed
        successfully.");
}
```

// Funcția de fitness

```
public void ComputeFitness(Chromosome c, int[,] data)
        {
            double sumAlfa = 0;

            int[] v1 = new int[6];
            int[] v2 = new int[6];

            for (int i = 0; i < c.NoGenes; ++i)
            {
                sumAlfa += c.Genes[i];
            }

            double sumAlfaiAlfaj = 0;
            for (int i = 0; i < c.NoGenes; ++i)
            {
                for (int j = 0; j < c.NoGenes; ++j)
                {
                    for (int ind1 = 0; ind1 < 6; ++ind1)
                    {
                        v1[ind1] = data[i, ind1];
                    }

                    for (int ind2 = 0; ind2 < 6; ++ind2)
                    {
                        v2[ind2] = data[j, ind2];
                    }
                    sumAlfaiAlfaj += c.Genes[i] * c.Genes[j] * data[i, 6] * data[j, 6] *
kernel1.PolinomialKernel(v1, v2);
                }
            }
            c.Fitness = sumAlfa - 0.5 * sumAlfaiAlfaj;
        }
```

// Determinarea soluției problemei – determinarea parametrilor alpha

```
Chromosome[] population = new Chromosome[populationSize];
            for (int i = 0; i < population.Length; i++)
            {
                population[i] = p.MakeChromosome();
                p.ComputeFitness(population[i], data);
            }

            for (int gen = 0; gen < maxGenerations; gen++)
            {
                Chromosome[] newPopulation = new Chromosome[populationSize];
                newPopulation[0] = Selection.GetBest(population); // elitism

                for (int i = 1; i < populationSize; i++)
                {
                    // Select 2 parents: Selection.Tournament
                    Chromosome p1 = new Chromosome(Selection.Tournament(population));
                    Chromosome p2 = new Chromosome(Selection.Tournament(population));

                    // Generating a child by crossover application: Crossover.Arithmetic
                    Chromosome c = new Chromosome(Crossover.Arithmetic(p1, p2, crossoverRate));

                    // Applying mutation to the child: Mutation.Reset
                    Mutation.Reset(c, mutationRate);

                    // Calculating the value of the fitness function for the child:
                    ComputeFitness from the problem p
                    p.ComputeFitness(c, data);

                    // Introduction of child in newPopulation
                    newPopulation[i] = c;
                }

                for (int i = 0; i < populationSize; i++)
                    population[i] = newPopulation[i];

                Console.Write("{0}% \r", (gen + 1));
            }
            Console.WriteLine();
            Console.WriteLine("Complete processing.");
            return Selection.GetBest(population);
```

// Funcția de calcul pentru nucleul polinomial de grad II

```
public int PolinomialKernel(int[] v1, int[] v2)
        {
            int result = 0;
            for (int i = 0; i < v1.Length; ++i)
            {
                result += (v1[i] * v2[i]);
            }
            result += 1;
            result *= result;

            return result;
        }
```

// **Script de ajustare a valorilor alpha pentru a respecta cele 2 condiții impuse de SVM**

```
int k;
double sPlus;
double sMinus;

    // Repair algorithm
    while (Math.Abs(s) > limitaSuma)
    {
        sPlus = 0.0;
        sMinus = 0.0;

        for (int i = 0; i < trainInstances; ++i)
        {
            if (i >= 91)
            {
                sPlus += newAlfa[i] * data[i, 6];
            }
            else
            {
                sMinus += newAlfa[i] * data[i, 6];
            }
        }
        sMinus = Math.Abs(sMinus);

        if (Math.Abs(sPlus) > Math.Abs(sMinus))
        {
            k = rand.Next(91, trainInstances);
        }
        else
        {
            k = rand.Next(0, 91);
        }

        if (newAlfa[k] > Math.Abs(s))
        {
            newAlfa[k] = newAlfa[k] - Math.Abs(s);
        }
        else
        {
            newAlfa[k] = 0;
        }

        s = 0;
        for (int i = 0; i < trainInstances; ++i)
        {
            s += newAlfa[i] * data[i, 6];
        }
    }
```

// Calcul Bias

```
    double bias = 0;
    double biasSum;

    int[] v1 = new int[6];
    int[] v2 = new int[6];

    for (int i = 0; i < trainInstances; ++i)
    {
        biasSum = 0;
        for (int j = 0; j < trainInstances; ++j)
        {
            for (int ind1 = 0; ind1 < 6; ++ind1)
            {
                v1[ind1] = data[i, ind1];
            }

            for (int ind2 = 0; ind2 < 6; ++ind2)
            {
                v2[ind2] = data[j, ind2];
            }

            biasSum += data[j, 6] * newAlfa[j] * kernel.PolinomialKernel(v1, v2);
        }
        bias += data[i, 6] - biasSum;
    }
    bias /= trainInstances;
```

// Calculare valoare de clasificare

```
Console.WriteLine("Calculation of classification value: ");
double[] classificationValue = new double[noInstances];

int[] trainV = new int[6];
int[] testV = new int[6];

double val;
for (int j = 0; j < noInstances; ++j)
{
       val = 0;
       for (int i = 0; i < trainInstances; ++i)
       {
              for (int ind1 = 0; ind1 < 6; ++ind1)
              {
                     trainV[ind1] = data[i, ind1];
              }

              for (int ind2 = 0; ind2 < 6; ++ind2)
              {
                     testV[ind2] = dataTest[j, ind2];
              }
              val += newAlfa[i] * data[i, 6] * kernel.PolinomialKernel(trainV, testV);
       }
       classificationValue[j] = val;
}
```

// **Determinarea claselor**

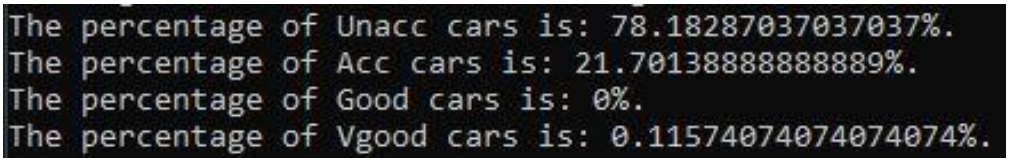
```
int countUnacc = 0;
int countAcc = 0;
int countGood = 0;
int countVgood = 0;

int[] classType = new int[noInstances];
for (int i = 0; i < noInstances; ++i)
{
	if (classifiedValues1[i] >= classifiedValues2[i] && classifiedValues1[i] >= 
classifiedValues3[i] && classifiedValues1[i] >= classifiedValues4[i])
	{
		classType[i] = 1;
		countUnacc++;
	}

	else if (classifiedValues2[i] >= classifiedValues1[i] && classifiedValues2[i] >= 
classifiedValues3[i] && classifiedValues2[i] >= classifiedValues4[i])
	{
		classType[i] = 2;
		countAcc++;
	}

	else if (classifiedValues3[i] >= classifiedValues1[i] && classifiedValues3[i] >= 
classifiedValues2[i] && classifiedValues3[i] >= classifiedValues4[i])
	{
		classType[i] = 3;
		countGood++;
	}

	else if (classifiedValues4[i] >= classifiedValues1[i] && classifiedValues4[i] >= 
classifiedValues2[i] && classifiedValues4[i] >= classifiedValues3[i])
	{
		classType[i] = 4;
		countVgood++;
	}
}
```

SECȚIUNEA 5: Rezultatele obținute prin rularea programului în diverse situații, capturi de ecran și comentarii asupra rezultatelor obținute

Mai jos, este prezentat rezultatul rulării algoritmului evolutiv pentru clasa acceptable. Se poate observa că numărul de zerouri este unul rezonabil, astfel numărul vectorilor suport va fi mai mic decât jumătatea numărului total de soluții alpha obținute. Ideal, ar fi fost un număr chiar mai redus de vectori suport. Se observă că valorile alpha-urilor respectă ambele condiții impuse de problema duală a SVM.

```
Acc class processing
The training data set was read successfully.
Alpha coefficients are calculated.
100%
Complete processing.

Lagrangian multipliers are: alpha = [6.776937212225549E-06 5.386700539564l1E-06 8.84070597069371E-06 6.363416494039555E-06 6.392991210517E-06 8.262091538990891E-06 5.105953139767961E-06 8.960249842591
917E-06 4.472186837565246E-07 3.866971807492426E-06 2.635408827399566E-06 1.446644701737286E-06 3.838967356755855E-06 2.471848578412026E-06 3.1790892841197E-06 9.629636821164586E-07 3.421817889167846E-0
6 7.970733981565916E-06 8.710636756713886E-06 6.658342716590666E-06 5.019867608798616E-07 2.728341888975516E-06 3.910136112901456E-06 3.511038093599976E-06 2.354956507848096E-06 7.489331374323726E-06 4.73
963540733775E-06 9.394626938456036E-06 9.885069127141076E-07 4.920285001825676E-06 2.899895810009866E-06 3.502712656512266E-06 3.998847740701886E-06 4.069104159283046E-06 7.593239689056416E-06 9.697396760
904743E-06 6.028226095264886E-06 9.708213624408576E-06 9.948148396773336E-06 3.706393960726636E-06 9.359100511930466E-06 5.664751583555426E-06 3.272103356789856E-06 7.376589461870776E-06 6.953151434172486
E-06 9.938372191944336E-06 1.650633770809856E-06 5.698711613984186E-06 9.151747156470916E-06 4.885900730679726E-06 9.191168378661936E-06 9.891703925045076E-06 8.514036502928496E-06 4.409836528082336E-06 7.51
708010561628E-06 4.048595323249986E-06 1.476564426662666E-06 5.329667877093736E-06 4.008459986191466E-06 8.903659833084636E-06 3.522456131653146E-06 5.711326201311936E-06 7.971742929877596E-06 5.526933
81930856E-06 9.670759974825556E-07 9.132696170887336E-06 7.447796961035496E-06 9.769863197472826E-06 4.878608889355616E-06 3.481692515072276E-06 9.934737586851586E-06 3.713604264759276E-06 8.022585584792
8E-06 5.004242795987176E-06 7.299242204660226E-06 4.837235177325666E-06 9.456238532185676E-06 6.533546651036686E-06 6.557886659427496E-06 2.416584315903766E-06 5.838697606622576E-06 9.040923793353576E-06
2.428797819851346E-07 6.102606182034416E-06 5.729116623163746E-06 3.008191884964796E-06 4.988676153583776E-06 9.589011212619496E-06 6.552024053666756E-06 3.544644365806436E-06 8.220832342384776E-06 6.985
582181711496E-07 5.699753638217116E-06 1.044488563688726E-06 1.839378290734976E-07 3.647774310618536E-06 7.693008653676616E-06 7.745169674859626E-06 6.409513464388216E-06 7.820455263285186E-06 ]

The initial value of the initial weighted sum of the coefficients is: -0.000305955037849934
The number of zeros is: 54
All Lagrangian coefficients fall within the limits imposed by the first condition of the dual problem, respectively fall within the range[0, 1E-05].
The first condition imposed by the dual problem is fulfilled.

The final value of the weighted sum of the coefficients is: 0
The second condition imposed by the dual problem is fulfilled.
The Lagrangian multipliers obtained after running the repair algorithm are: alpha = [0 0 0 0 6.392991210517E-06 8.262091538990889E-06 5.105953139767961E-06 8.960249842591E-06 0 0 0 0 3.838967356
75585E-06 0 0 9.629636821164586E-07 0 0 8.710636756713786E-06 0 5.019867608798616E-07 2.728341888975516E-06 3.910136112901456E-06 3.511038093599976E-06 0 0 0 9.394626938456036E-06 9.885069127141076E-07 4
.920285001825676E-06 0 0 0 0 4.069104159283046E-06 0 7.467818775897636E-07 6.028226095264886E-06 0 0 0 0 5.664751583555426E-06 0 0 0 9.938372191944336E-06 1.650633770809856E-06 0 0 0 0 0 0 0 0 4.0485953232
4998E-06 0 0 4.008459986191456E-06 0 0 0 7.971742929877596E-06 5.526933681930856E-06 9.670759974825556E-07 9.132696170887336E-06 7.447796961035496E-06 9.769863197472826E-06 4.878608889355596E-06 3.481692
51507227E-06 4.934737586851586E-06 3.713604264759276E-06 8.022585584792586E-06 5.004242795987176E-06 7.299242204660226E-06 4.837235177325666E-06 9.456238532185676E-06 6.533546651036686E-06 6.557886659427
49E-06 2.416584315903766E-06 5.838697606622576E-06 9.040923793353576E-06 0 0 0 0 0 0 6.552024053666756E-06 3.544644365806426E-06 0 6.985582181711496E-07 0 0 0 0 7.693008653676616E-06 0 0 0 ]

Bias value is: -0.6
////////////////////////////////////////////////   Processing completed for Acc class   ////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

The test dataset was read successfully.


Calculation of classification value:

The values in the vector with classification value are:
The string obtained is:
-0.035113522083411 6,-0.0327356418484196,-0.0327356418484196,-0.0320266558035168,-0.0292511617111,-0.0292511617111,-0.028426522816553,-0.0252534148667116,-0.0252534148667116,-0.030815045488358,-0
.027903748052270 9,-0.0279037480522709,-0.0270691819012208,-0.023760270607709,-0.023760270607709,-0.0228100516070146,-0.0191035264560782,-0.0191035264560782,-0.0259324267293849,-0.022487712092203
,-0.022487712092203,-0.0215275658350054,-0.0176852373403986,-0.0176852373403986,-0.0166094382335568,-0.0123694958815255,-0.0123694958815255,-0.0350128184256064,-0.0324105502466813,-0.032410550246
66813,-0.0316270660093831,-0.0286271839730335,-0.0286271839730335,-0.0277280468860911,-0.0243305509923167,-0.0243305509923167,-0.0303724725504 65,-0.0272367871704449,-0.0272367871704449,-0.026327
7228269994,-0.0227944235895548,-0.0227944235895548,-0.021769706396465,-0.0178387933015957,-0.0178387933015957,-0.0251479845114043,-0.0214788819302892,-0.0214788819302892,-0.0204442374806963,-0.0
163775210421567,-0.0163775210421567,-0.0152272237429195,-0.0107628934469553,-0.0107628934469553,-0.0347837393030355,-0.0319570831801775,-0.0319570831801775,-0.0310991007504841,-0.027874830770201
4,-0.0278748307702014,-0.0269011954908637,-0.0232793116531564,-0.0232793116531564,-0.0298015241478064,-0.0264414508238534,-0.0264414508238534,-0.0254578882880126,-0.0217002011066635,-0.0217002011
06635,-0.0206009857211499,-0.0164456846823476,-0.0164456846823476,-0.0242351668286858,-0.02034167630361,-0.0203416763036 1,-0.0192325336616218,-0.0149414292791491,-0.0149414292791491,-0.0137166337
875168,-0.0090279155476194,-0.0090279155476194,-0.0344262847156991,-0.0313752406489082,-0.0313752406489082,-0.0304427600268194,-0.0269941021026037,-0.0269941021026037,-0.0259459686308707,-0.0220
996968492305,-0.0220996968492305,-0.0291022002803823,-0.0255177390124963,-0.0255177390124963,-0.0244596782842602,-0.0204776031589496,-0.0204776031589496,-0.0193038895810692,-0.0149242005983 34,-0
.014924200598334,-0.0231939736811463,-0.0190760952121653,-0.0190760952121653,-0.0178924543777818,-0.0133769620513761,-0.0133769620513761,-0.0120776683673484,-0.0071645621835 1816,-0.0071645621835
```

Rezultatele obținute în urma rulării script-ului de clasificare se pot observa în imaginea de mai jos. Este posibil ca o serie de modificări asupra parametrilor SVM-ului să conducă la rezultate mai precise.



SECȚIUNEA 6: Concluzii

În concluzie, SVM reprezintă o metodă precisă de clasificare, însă dificultatea acestui model apare dintr-o serie de motive precum: alegerea potrivită a tipurilor parametrilor (nucleu), dar și a valorilor acestora (cost), precum și ajustarea seturilor de date utlizate pentru antrenare și testare, astfel încât acestea să respecte aceeași distribuție de probabilitate. Așadar, plecând de la teoria de învățare statistică în încercarea de a rezolva o problemă într-un mod riguros explicit, se obțin rezultate bune și din punct de vedere practic.

SECȚIUNEA 7: Bibliografie

1. https://archive.ics.uci.edu/ml/index.php
2. https://archive.ics.uci.edu/ml/datasets/Car+Evaluation
3. Evolutionary Support Vector Machines: A Dual Approach - Madson Luiz Dantas Dias, Ajalmar R. Rocha Neto - Department of Teleinformatics, Federal Institute of Ceará