using FuzzyCoreUtils;

using FuzzySystem.FuzzyAbstract;

using FuzzySystem.FuzzyAbstract.conf;

using FuzzySystem.FuzzyAbstract.learn\_algorithm.conf;

using System;

using System.Collections.Generic;

using System.Linq;

namespace FuzzySystem.TakagiSugenoApproximate.LearnAlgorithm

{

public class BSapprox : AbstractNotSafeLearnAlgorithm

{

List<int[]> groups;

protected Random rand = new Random();

protected TSAFuzzySystem result;

protected BSConfig config;

protected KnowlegeBaseTSARules[] Population;

protected KnowlegeBaseTSARules[] NewPopulation;

protected int N, m, iter, cur\_iter;

protected double p\_one, p\_one\_center, p\_two\_center, F;

protected int[] NS;

public override TSAFuzzySystem TuneUpFuzzySystem(TSAFuzzySystem Approx, ILearnAlgorithmConf conf)

{

result = Approx;

groups = new List<int[]>();

Init(conf);

SetPopulation();

Population = ListTakagiSugenoApproximateTool.SortRules(Population, result);

NS = new int[m];

for (int i = 0; i < m; i++)

{

NS[i] = (N - 1) / m;

}

cur\_iter = 0;

while (cur\_iter < iter)

{

groups = GroupStream();

if (p\_one > rand.NextDouble())

{

ChooseOneCluster();

}

else

{

ChooseTwoClusters();

}

Population = ListTakagiSugenoApproximateTool.SortRules(Population, result);

Console.WriteLine(cur\_iter + " - Итерация");

Console.WriteLine("Обуч. выборка = " + result.ErrorLearnSamples(Population[0]));

Console.WriteLine("Тест. выборка = " + result.ErrorTestSamples(Population[0]));

cur\_iter++;

}

Population = ListTakagiSugenoApproximateTool.SortRules(Population, result);

result.RulesDatabaseSet[0] = Population[0];

return result;

}

public virtual void Init(ILearnAlgorithmConf conf)

{

config = conf as BSConfig;

iter = ((BSConfig)conf).iter;

N = ((BSConfig)conf).N;

m = ((BSConfig)conf).m;

F = ((BSConfig)conf).F;

p\_one = ((BSConfig)conf).p\_one;

p\_one\_center = ((BSConfig)conf).p\_one\_center;

p\_two\_center = ((BSConfig)conf).p\_two\_center;

}

private void SetPopulation()

{

Population = new KnowlegeBaseTSARules[N];

KnowlegeBaseTSARules TempRule = new KnowlegeBaseTSARules(result.RulesDatabaseSet[0]);

Population[0] = TempRule;

for (int i = 1; i < N; i++)

{

TempRule = new KnowlegeBaseTSARules(result.RulesDatabaseSet[0]);

Population[i] = TempRule;

for (int j = 0; j < Population[i].TermsSet.Count; j++)

{

for (int k = 0; k < Population[i].TermsSet[j].Parametrs.Length; k++)

{

Population[i].TermsSet[j].Parametrs[k] = GaussRandom.Random\_gaussian(rand, Population[i].TermsSet[j].Parametrs[k], 0.1 \* Population[i].TermsSet[j].Parametrs[k]);

}

}

}

}

private List<int[]> GroupStream()

{

List<int[]> GroupsCalc = new List<int[]>();

Dictionary<int, double> distances = new Dictionary<int, double>();

for (int j = 1; j < N; j++)

{

distances.Add(j, 1);

}

for (int i = 0; i < m; i++)

{

int[] group = new int[NS[i]];

foreach (int j in distances.Keys.ToArray())

{

distances[j] = Distance(Population[0], Population[j]);

}

for (int j = 0; j < NS[i]; j++)

{

var KeyMinValue = GetKeyByValue(distances, distances.Values.Min());

group[j] = KeyMinValue;

distances.Remove(KeyMinValue);

}

GroupsCalc.Add(group);

}

return GroupsCalc;

}

private static int GetKeyByValue(Dictionary<int, double> myDictionary, double value)

{

foreach (var recordOfDictionary in myDictionary)

{

if (recordOfDictionary.Value.Equals(value))

return recordOfDictionary.Key;

}

return -1;

}

private double Distance(KnowlegeBaseTSARules x, KnowlegeBaseTSARules y)

{

double dist, sum = 0;

for (int i = 0; i < x.TermsSet.Count; i++)

{

for (int j = 0; j < x.TermsSet[j].Parametrs.Length; j++)

{

sum += Math.Pow(x.TermsSet[i].Parametrs[j] + y.TermsSet[i].Parametrs[j], 2);

}

}

for (int i = 0; i < x.RulesDatabase.Count; i++)

{

sum += Math.Pow(x.RulesDatabase[i].Cons\_DoubleOutput + y.RulesDatabase[i].Cons\_DoubleOutput, 2);

}

dist = Math.Sqrt(sum);

return dist;

}

private void ChooseOneCluster()

{

int cluster\_index = rand.Next(0, m);

if (p\_one\_center > rand.NextDouble())

{

Console.WriteLine("!");

OriginalOperator(cluster\_index);

}

else

{

Console.WriteLine("!!");

OneDEOperator(cluster\_index);

}

}

private void OriginalOperator(int cluster\_index)

{

NewPopulation = new KnowlegeBaseTSARules[groups[cluster\_index].Length];

double epsi\_newstep = rand.NextDouble() \* Math.Exp(1 - (iter / (iter - cur\_iter + 1)));

for (int i = 0; i < groups[cluster\_index].Length; i++)

{

NewPopulation[i] = Population[groups[cluster\_index][i]];

}

for (int i = 0; i < groups[cluster\_index].Length; i++)

{

int number = groups[cluster\_index][i];

for (int j = 0; j < NewPopulation[i].TermsSet.Count; j++)

{

for (int k = 0; k < NewPopulation[i].TermsSet[j].Parametrs.Length; k++)

{

NewPopulation[i].TermsSet[j].Parametrs[k] += epsi\_newstep \* rand.NextDouble();

}

}

if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number]))

{

Population[number] = NewPopulation[i];

}

}

}

private void OneDEOperator(int cluster\_index)

{

NewPopulation = new KnowlegeBaseTSARules[groups[cluster\_index].Length];

NewPopulation[0] = Population[0];

double epsi = rand.NextDouble() \* Math.Exp(1 - (iter / (iter - cur\_iter + 1)));

for (int i = 1; i < groups[cluster\_index].Length; i++)

{

NewPopulation[i] = Population[groups[cluster\_index][i]];

}

for (int i = 1; i < groups[cluster\_index].Length; i++)

{

int rand1 = rand.Next(0, groups[cluster\_index].Length);

int rand2 = rand.Next(0, groups[cluster\_index].Length);

int rand3 = rand.Next(0, groups[cluster\_index].Length);

while (rand2 == rand1)

rand2 = rand.Next(0, groups[cluster\_index].Length);

while ((rand3 == rand1) || (rand3 == rand2))

rand3 = rand.Next(0, groups[cluster\_index].Length);

int number1 = groups[cluster\_index][rand1];

int number2 = groups[cluster\_index][rand2];

int number3 = groups[cluster\_index][rand3];

for (int j = 0; j < Population[groups[cluster\_index][i]].TermsSet.Count; j++)

{

for (int k = 0; k < Population[groups[cluster\_index][i]].TermsSet[j].Parametrs.Length; k++)

{

NewPopulation[i].TermsSet[j].Parametrs[k] = Population[number1].TermsSet[j].Parametrs[k] + F \* (Population[number2].TermsSet[j].Parametrs[k] - Population[number3].TermsSet[j].Parametrs[k]);

}

}

if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number1]))

{

Population[number1] = NewPopulation[i];

}

else

{

if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number2]))

{

Population[number2] = NewPopulation[i];

}

else if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number3]))

{

Population[number3] = NewPopulation[i];

}

}

for (int j = 0; j < Population[groups[cluster\_index][i]].TermsSet.Count; j++)

{

for (int k = 0; k < Population[groups[cluster\_index][i]].TermsSet[j].Parametrs.Length; k++)

{

NewPopulation[i].TermsSet[j].Parametrs[k] += epsi \* rand.NextDouble();

}

}

if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number1]))

{

Population[number1] = NewPopulation[i];

}

else

{

if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number2]))

{

Population[number2] = NewPopulation[i];

}

else if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number3]))

{

Population[number3] = NewPopulation[i];

}

}

}

}

private void ChooseTwoClusters()

{

if (p\_two\_center > rand.NextDouble())

{

Console.WriteLine("!!!");

OriginalTwoClusters();

}

else

{

Console.WriteLine("!!!!");

TwoDEOperator();

}

}

private void OriginalTwoClusters()

{

int cluster\_index\_1 = rand.Next(0, m);

int cluster\_index\_2 = rand.Next(0, m);

while (cluster\_index\_1 == cluster\_index\_2)

cluster\_index\_2 = rand.Next(0, m);

NewPopulation = new KnowlegeBaseTSARules[groups[cluster\_index\_1].Length];

double epsi = rand.NextDouble() \* Math.Exp(1 - (iter / (iter - cur\_iter + 1)));

for (int i = 0; i < groups[cluster\_index\_1].Length; i++)

{

NewPopulation[i] = Population[groups[cluster\_index\_1][i]];

}

for (int i = 0; i < groups[cluster\_index\_1].Length; i++)

{

int number1 = groups[cluster\_index\_1][i];

int number2 = groups[cluster\_index\_2][i];

for (int j = 0; j < Population[groups[cluster\_index\_1][i]].TermsSet.Count; j++)

{

for (int k = 0; k < Population[groups[cluster\_index\_1][i]].TermsSet[j].Parametrs.Length; k++)

{

double rand1 = rand.NextDouble();

NewPopulation[i].TermsSet[j].Parametrs[k] = rand1 \* Population[number1].TermsSet[j].Parametrs[k] + (1 - rand1) \* Population[number2].TermsSet[j].Parametrs[k];

}

}

if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number1]))

{

Population[number1] = NewPopulation[i];

}

else if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number2]))

{

Population[number2] = NewPopulation[i];

}

for (int j = 0; j < Population[groups[cluster\_index\_1][i]].TermsSet.Count; j++)

{

for (int k = 0; k < Population[groups[cluster\_index\_1][i]].TermsSet[j].Parametrs.Length; k++)

{

NewPopulation[i].TermsSet[j].Parametrs[k] += epsi \* rand.NextDouble();

}

}

if(result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number1]))

{

Population[number1] = NewPopulation[i];

}

else if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number2]))

{

Population[number2] = NewPopulation[i];

}

}

}

private void TwoDEOperator()

{

int cluster\_index\_1 = rand.Next(0, m);

int cluster\_index\_2 = rand.Next(0, m);

while (cluster\_index\_1 == cluster\_index\_2)

cluster\_index\_2 = rand.Next(0, m);

NewPopulation = new KnowlegeBaseTSARules[groups[cluster\_index\_1].Length];

NewPopulation[0] = Population[0];

double epsi = rand.NextDouble() \* Math.Exp(1 - (iter / (iter - cur\_iter + 1)));

for (int i = 1; i < groups[cluster\_index\_1].Length; i++)

{

NewPopulation[i] = Population[groups[cluster\_index\_1][i]];

}

for (int i = 1; i < groups[cluster\_index\_1].Length; i++)

{

int rand1 = rand.Next(0, groups[cluster\_index\_1].Length);

int rand2 = rand.Next(0, groups[cluster\_index\_2].Length);

while (rand2 == rand1)

rand2 = rand.Next(0, groups[cluster\_index\_2].Length);

int number1 = groups[cluster\_index\_1][rand1];

int number2 = groups[cluster\_index\_2][rand2];

for (int j = 0; j < Population[groups[cluster\_index\_1][i]].TermsSet.Count; j++)

{

for (int k = 0; k < Population[groups[cluster\_index\_1][i]].TermsSet[j].Parametrs.Length; k++)

{

NewPopulation[i].TermsSet[j].Parametrs[k] = Population[0].TermsSet[j].Parametrs[k] + F \* (Population[number1].TermsSet[j].Parametrs[k] - Population[number2].TermsSet[j].Parametrs[k]);

}

}

if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number1]))

{

Population[number1] = NewPopulation[i];

}

else if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number2]))

{

Population[number2] = NewPopulation[i];

}

for (int j = 0; j < Population[groups[cluster\_index\_1][i]].TermsSet.Count; j++)

{

for (int k = 0; k < Population[groups[cluster\_index\_1][i]].TermsSet[j].Parametrs.Length; k++)

{

NewPopulation[i].TermsSet[j].Parametrs[k] += epsi \* rand.NextDouble();

}

}

if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number1]))

{

Population[number1] = NewPopulation[i];

}

else if (result.approxLearnSamples(NewPopulation[i]) < result.approxLearnSamples(Population[number2]))

{

Population[number2] = NewPopulation[i];

}

}

}

public override List<FuzzySystemRelisedList.TypeSystem> SupportedFS

{

get

{

return new List<FuzzySystemRelisedList.TypeSystem>()

{

FuzzySystemRelisedList.TypeSystem.TakagiSugenoApproximate

};

}

}

public override ILearnAlgorithmConf getConf(int CountFeatures)

{

BSConfig conf = new BSConfig();

conf.Init(CountFeatures);

return conf;

}

public override string ToString(bool with\_param = false)

{

if (with\_param)

{

string result = "Brain Storm Algorithm" + "{" + Environment.NewLine;

result = "Итераций = " + iter + ";" + Environment.NewLine;

result = "Идей = " + N + ";" + Environment.NewLine;

return result;

}

return "Brain Storm Algorithm";

}

}

}