# ПРИЛОЖЕНИЕ А

*(справочное)*

Листинг кода класса FuzzyPredictor

public class FuzzyPredictor : IPredictor

{

private List<MatchInfo> matchesInfo;

public FuzzyPredictor(List<MatchInfo> \_matchesInfo)

{

matchesInfo = \_matchesInfo;

}

public double[] predict(MatchInfo matchInfo)

{

string[] matchesFuzzyInfo = new string[matchesInfo.Count];

int i = 0;

string matchFuzzyInfo = "";

foreach (MatchInfo item in matchesInfo)

{

matchesFuzzyInfo[i++] = item.firstTeam.fuzzyInfo + item.secondTeam.fuzzyInfo;

if (item.id == matchInfo.id)

matchFuzzyInfo = matchesFuzzyInfo[i - 1];

}

int[] maxSimilarMatchesIds = new int[] { 0, 0, 0 };

int[] similarCounts = new int[] { 0, 0, 0 };

int currentSimilar;

int matchResultType;

for (i = 0; i < matchesInfo.Count; i++)

{

if (matchesInfo[i].id == matchInfo.id || matchesInfo[i].realResult == 3)

continue;

currentSimilar = 0;

matchResultType = matchesInfo[i].realResult;

for (int j = 0; j < matchFuzzyInfo.Length; j++)

{

if (matchFuzzyInfo[j] == matchesFuzzyInfo[i][j])

currentSimilar++;

if (currentSimilar > similarCounts[matchResultType])

{

similarCounts[matchResultType] = currentSimilar;

maxSimilarMatchesIds[matchResultType] = i;

}

}

}

double[] prediction = new double[] {

similarCounts[0] / matchFuzzyInfo.Length,

similarCounts[1] / matchFuzzyInfo.Length,

similarCounts[2] / matchFuzzyInfo.Length,

};

return prediction;

}

public static string analizeTeamInfo(List<Parameter> teamParameters)

{

string result = "";

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

{

double value = teamParameters[i].value;

int leftLimit = teamParameters[i].fuzzyLimit.leftLimit;

int rigtLimit = teamParameters[i].fuzzyLimit.rightLimit;

if (value < leftLimit)

result += "1";

else if (value > rigtLimit)

result += "3";

else

result += "2";

}

return result;

}

}

# ПРИЛОЖЕНИЕ Б

*(справочное)*

Листинг кода класса GeneralPredictor

public class GeneralPredictor : IPredictor

{

private QualimetricPredictor qualimetricPredictor;

private FuzzyPredictor fuzzyPredictor;

private NeuralNetworkPredictor neuralNetworkPredictor;

public GeneralPredictor(List<MatchInfo> matchesInfo, List<OutputLayerWeights> outputLayerWeights, List<TeachedRBF> teachedRBFs)

{

qualimetricPredictor = new QualimetricPredictor();

fuzzyPredictor = new FuzzyPredictor(matchesInfo);

neuralNetworkPredictor = new NeuralNetworkPredictor(matchesInfo, outputLayerWeights, teachedRBFs, false);

}

public double[] predict(MatchInfo matchInfo)

{

double[] qualimetricPrediction = qualimetricPredictor.predict(matchInfo);

double[] fuzzyPrediction = fuzzyPredictor.predict(matchInfo);

double[] neuralNetworkPrediction = neuralNetworkPredictor.predict(matchInfo);

double[] finalPrediction = new double[3];

for (int i = 0; i < 3; i++)

{

finalPrediction[i] = neuralNetworkPrediction[i];

finalPrediction[i] += fuzzyPrediction[i] / 3;

finalPrediction[i] += qualimetricPrediction[i] \* 0.3;

if (finalPrediction[i] < 0.15)

finalPrediction[i] = 0.15 + i / 100;

}

double sum = finalPrediction.Sum();

for (int i = 0; i < 3; i++)

finalPrediction[i] /= sum;

return finalPrediction;

}

}

# ПРИЛОЖЕНИЕ В

*(справочное)*

Листинг кода класса QualimetricPredictor

public class QualimetricPredictor : IPredictor

{

public double[] predict(MatchInfo matchInfo)

{

double[] teamPowers = TeamPowerCalculator.calculateTeamPowers(matchInfo.firstTeam.getListOfParameters(), matchInfo.secondTeam.getListOfParameters());

double[] prediction = new double[] {

teamPowers.Min() / teamPowers.Max(),

teamPowers[0],

teamPowers[1]

};

return prediction;

}

}

# ПРИЛОЖЕНИЕ Г

*(справочное)*

Листинг кода класса TeamPowerCalculator

public static class TeamPowerCalculator

{

public static double[] calculateTeamPowers(List<Parameter> firstTeamParameters, List<Parameter> secondTeamParameters)

{

double[] teamPowers = new double[2] { 0, 0 };

double[] teamsRelativeParemeters = new double[2];

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

{

teamsRelativeParemeters = calculateRelativeParameters(firstTeamParameters[i], secondTeamParameters[i]);

teamPowers[0] += teamsRelativeParemeters[0] \* firstTeamParameters[i].weight;

teamPowers[1] += teamsRelativeParemeters[1] \* secondTeamParameters[i].weight;

}

double teamPowersSum = teamPowers[0] + teamPowers[1];

teamPowers[0] /= teamPowersSum;

teamPowers[1] /= teamPowersSum;

return teamPowers;

}

static double[] calculateRelativeParameters(Parameter firstTeamParemeter, Parameter secondTeamParameter)

{

double[] relativeParameters = new double[2];

double sum = firstTeamParemeter.value + secondTeamParameter.value;

if (sum == 0)

{

relativeParameters[0] = 0;

relativeParameters[1] = 0;

}

else

{

double firstTeamRelativeParemeter = firstTeamParemeter.value / sum;

if (!firstTeamParemeter.isStimulator)

firstTeamRelativeParemeter = 1 - firstTeamRelativeParemeter;

double secondTeamRelativeParemeter = secondTeamParameter.value / sum;

if (!secondTeamParameter.isStimulator)

secondTeamRelativeParemeter = 1 - firstTeamRelativeParemeter;

relativeParameters[0] = firstTeamRelativeParemeter;

relativeParameters[1] = secondTeamRelativeParemeter;

}

return relativeParameters;

}

}

# ПРИЛОЖЕНИЕ Д

*(справочное)*

Листинг кода класса Serializer

public static class Serializer

{

public static string SerializeToString(double[] array)

{

string result = "";

foreach (double item in array)

{

if (result.Length > 0)

result += '|';

result += item.ToString();

}

return result;

}

public static double[] DeserializeFromString(string serializedArray)

{

string[] deserializedArray = serializedArray.Split('|');

double[] result = new double[deserializedArray.Length];

int i = 0;

foreach (string item in deserializedArray)

{

result[i++] = double.Parse(item, CultureInfo.InvariantCulture);

}

return result;

}

}

# ПРИЛОЖЕНИЕ Е

*(справочное)*

Листинг кода класса NeuralNetworkPredictor

public class NeuralNetworkPredictor : IPredictor

{

private static RadialBasisFunctionNetwork rbfNetwork;

private static bool \_isNetworkChanged;

public NeuralNetworkPredictor(List<MatchInfo> matchesInfo, List<OutputLayerWeights> outputLayerWeights, List<TeachedRBF> teachedRBFs, bool isTeachNeaded)

{

\_isNetworkChanged = false;

initializeNeuralNetwork(matchesInfo, outputLayerWeights, teachedRBFs, isTeachNeaded);

}

private void initializeNeuralNetwork(List<MatchInfo> matchesInfo, List<OutputLayerWeights> outputLayerWeights, List<TeachedRBF> teachedRBFs, bool isTeachNeaded)

{

if (outputLayerWeights.Count > 0 && outputLayerWeights.Count == teachedRBFs.Count && isTeachNeaded != true)

{

rbfNetwork = new RadialBasisFunctionNetwork(matchesInfo[0].firstTeam.getListOfParameters().Count \* 2,

3, teachedRBFs.Count, teachedRBFs, outputLayerWeights,

new char[] { '0', '1', '2' });

}

else

{

Dictionary<char, List<double[]>> classes = new Dictionary<char, List<double[]>>();

List<double[]> zeroMatches = new List<double[]>();

List<double[]> vinMatches = new List<double[]>();

List<double[]> looseMatches = new List<double[]>();

foreach (MatchInfo matchInfo in matchesInfo)

{

if (matchInfo.realResult == 3)

continue;

double[] matchParams = matchInfo.getMatchParams();

if (matchInfo.realResult == 0)

zeroMatches.Add(matchParams);

else if (matchInfo.realResult == 1)

vinMatches.Add(matchParams);

else

looseMatches.Add(matchParams);

}

classes['0'] = zeroMatches;

classes['1'] = vinMatches;

classes['2'] = looseMatches;

while (true)

{

rbfNetwork = new RadialBasisFunctionNetwork(classes, 1, 0.1);

if (rbfNetwork.iterationCount != 1000000)

break;

}

\_isNetworkChanged = true;

}

}

public static void prepareDataForSaving(List<OutputLayerWeights> outputLayerWeights, List<TeachedRBF> teachedRBFs)

{

rbfNetwork.prepareDataForSaving(outputLayerWeights, teachedRBFs);

}

public double[] predict(MatchInfo matchInfo)

{

Dictionary<char, double> prediction = rbfNetwork.ClassifyMatch(matchInfo.getMatchParams());

double[] convertedPrediction = new double[] {

prediction['0'],

prediction['1'],

prediction['2'],

};

return convertedPrediction;

}

public static bool isNetworkChanged()

{

return \_isNetworkChanged;

}

}

ПРИЛОЖЕНИЕ Ж

*(справочное)*

Листинг кода класса RadialBasisFunctionNetwork

public class RadialBasisFunctionNetwork

{

private RBF\_teachInfo[] rbf\_teached;

private double[,] outputLayer\_weights;

private int inputCount;

private int outputCount;

private int rbfCount;

private double alfa;

public double maxErrorTreshold { get; private set; }

public int iterationCount { get; private set; }

private char[] classesNames;

public RadialBasisFunctionNetwork(int \_inputCount, int \_outputCount, int \_rbfCount, List<TeachedRBF> \_teachedRBFs, List<OutputLayerWeights> \_outputLayerWeights, char[] \_classesNames)

{

inputCount = \_inputCount;

outputCount = \_outputCount;

rbfCount = \_rbfCount;

rbf\_teached = new RBF\_teachInfo[rbfCount];

outputLayer\_weights = new double[rbfCount, outputCount];

int i = 0;

foreach (TeachedRBF teachedRBF in \_teachedRBFs)

{

RBF\_teachInfo rbf = new RBF\_teachInfo();

rbf.standardDeviation = teachedRBF.StandartDeviation;

rbf.expectations = Serializer.DeserializeFromString(teachedRBF.Expectation);

rbf\_teached[i++] = rbf;

}

i = 0;

outputLayer\_weights = new double[rbfCount, outputCount];

for (i = 0; i < rbfCount; i++)

{

for (int j = 0; j < outputCount; j++)

{

outputLayer\_weights[i, j] = Serializer.DeserializeFromString(\_outputLayerWeights[i].OutputLayerWeight)[j];

}

}

classesNames = \_classesNames;

iterationCount = 0;

}

public RadialBasisFunctionNetwork(Dictionary<char, List<double[]>> classes, double alfa, double maxError)

{

int size = classes.First().Value.First().Length;

int i = 0;

classesNames = new char[classes.Count];

foreach (KeyValuePair<char, List<double[]>> tmp in classes)

classesNames[i++] = tmp.Key;

this.alfa = alfa;

this.maxErrorTreshold = maxError;

iterationCount = TeachNeuralNetwork(classes);

}

private int TeachNeuralNetwork(Dictionary<char, List<double[]>> classes)

{

inputCount = classes.First().Value.First().Length;

outputCount = classes.Count;

rbfCount = outputCount;

TeachRBFNeurons(classes);

double[][][] rbfNeurons = CalculateRBFNeuronsForTrainingSet(classes);

outputLayer\_weights = new double[rbfCount, outputCount];

Randomize();

double[] outputNeurons = new double[outputCount];

List<double> maxErrors = new List<double>();

int iterCount = 0;

double errorSumPrev = double.MaxValue;

double[] outputErrors = new double[outputCount];

double[] outputErrorsAbs = new double[outputCount];

double mu = 1;

while (true)

{

maxErrors.Clear();

int classNumber = 0;

foreach (List<double[]> matches in classes.Values)

{

int matchNumber = 0;

foreach (double[] match in matches)

{

for (int k = 0; k < outputCount; k++)

{

double sum = 0;

for (int j = 0; j < rbfCount; j++)

sum += outputLayer\_weights[j, k] \* rbfNeurons[classNumber][matchNumber][j];

outputNeurons[k] = sum;

}

Array.Clear(outputErrors, 0, outputErrors.Length);

outputErrors[classNumber] = 1;

for (int k = 0; k < outputCount; k++)

outputErrors[k] -= outputNeurons[k];

outputErrorsAbs = outputErrors.Select(x => Math.Abs(x)).ToArray();

maxErrors.Add(outputErrorsAbs.Max());

for (int j = 0; j < rbfCount; j++)

{

for (int k = 0; k < outputCount; k++)

{

outputLayer\_weights[j, k] = mu \* outputLayer\_weights[j, k] +

alfa \* outputErrors[k] \* rbfNeurons[classNumber][matchNumber][j];

}

}

matchNumber++;

}

classNumber++;

}

iterCount++;

if (iterCount == 1000000)

break;

double maxError = maxErrors.Max();

if (maxError < maxErrorTreshold)

break;

double errorSum = outputErrorsAbs.Sum();

if (errorSum >= errorSumPrev)

{

Randomize();

errorSumPrev = double.MaxValue;

}

else

errorSumPrev = errorSum;

}

return iterCount;

}

private void TeachRBFNeurons(Dictionary<char, List<double[]>> classes)

{

List<RBF\_teachInfo> RBFs = new List<RBF\_teachInfo>();

foreach (List<double[]> matchClass in classes.Values)

{

foreach (double[] match in matchClass)

{

RBF\_teachInfo rbf = new RBF\_teachInfo();

rbf.expectations = match;

RBFs.Add(rbf);

}

}

rbfCount = RBFs.Count;

rbf\_teached = RBFs.ToArray();

if (rbfCount == 1)

{

double sko = 0;

for (int k = 0; k < inputCount; k++)

sko += Math.Pow((rbf\_teached[0].expectations[k]), 2);

rbf\_teached[0].standardDeviation = Math.Sqrt(sko);

return;

}

List<double>[] deviations = new List<double>[rbfCount];

for (int i = 0; i < deviations.Length; i++)

deviations[i] = new List<double>();

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

{

for (int j = i + 1; j < rbfCount; j++)

{

double sko = 0;

for (int k = 0; k < inputCount; k++)

sko += Math.Pow((rbf\_teached[i].expectations[k] - rbf\_teached[j].expectations[k]), 2);

sko = Math.Sqrt(sko);

deviations[i].Add(sko);

deviations[j].Add(sko);

}

}

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

rbf\_teached[i].standardDeviation = deviations[i].Min();

}

private double[][][] CalculateRBFNeuronsForTrainingSet(Dictionary<char, List<double[]>> classes)

{

double[][][] result = new double[classes.Count][][];

int classCount = 0;

foreach (List<double[]> matchClass in classes.Values)

{

result[classCount] = new double[matchClass.Count][];

int matchCount = 0;

foreach (double[] match in matchClass)

{

result[classCount][matchCount] = new double[rbfCount];

for (int j = 0; j < rbfCount; j++)

{

double sum = 0;

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

sum += (match[i] - rbf\_teached[j].expectations[i]) \* (match[i] - rbf\_teached[j].expectations[i]);

result[classCount][matchCount][j] = Math.Exp(-sum / Math.Pow(rbf\_teached[j].standardDeviation, 2));

}

matchCount++;

}

classCount++;

}

return result;

}

private void Randomize()

{

Random random = new Random(RandomProvider.Next() ^ Environment.TickCount);

for (int i = 0; i < outputLayer\_weights.GetLength(0); i++)

{

for (int j = 0; j < outputLayer\_weights.GetLength(1); j++)

outputLayer\_weights[i, j] = 2 \* random.NextDouble() - 1;

}

}

public Dictionary<char, double> ClassifyMatch(double[] match)

{

if (inputCount != match.Length)

throw new Exception("Match has wrong size.");

double[] rbfNeurons = new double[rbfCount];

double[] outputNeurons = new double[outputCount];

for (int j = 0; j < rbfCount; j++)

{

double sum = 0;

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

sum += (match[i] - rbf\_teached[j].expectations[i]) \* (match[i] - rbf\_teached[j].expectations[i]);

rbfNeurons[j] = Math.Exp(-sum / Math.Pow(rbf\_teached[j].standardDeviation, 2));

}

for (int k = 0; k < outputCount; k++)

{

double sum = 0;

for (int j = 0; j < rbfCount; j++)

sum += outputLayer\_weights[j, k] \* rbfNeurons[j];

if (sum < 0)

sum = 0;

outputNeurons[k] = sum;

}

Dictionary<char, double> result = new Dictionary<char, double>();

int t = 0;

foreach (char className in classesNames)

{

result.Add(classesNames[t], outputNeurons[t]);

t++;

}

return result;

}

public void prepareDataForSaving(List<OutputLayerWeights> outputLayerWeights, List<TeachedRBF> teachedRBFs)

{

int i = 1;

foreach (var item in rbf\_teached)

{

TeachedRBF teachedRBF = new TeachedRBF();

teachedRBF.Id = i++;

teachedRBF.StandartDeviation = item.standardDeviation;

teachedRBF.Expectation = Serializer.SerializeToString(item.expectations);

teachedRBFs.Add(teachedRBF);

}

for (i = 0; i < rbfCount; i++)

{

OutputLayerWeights outputLayerWeightsItem = new OutputLayerWeights();

outputLayerWeightsItem.Id = i + 1;

outputLayerWeightsItem.OutputLayerWeight = Serializer.SerializeToString(

new double[] { outputLayer\_weights[i, 0],

outputLayer\_weights[i, 1],

outputLayer\_weights[i, 2] });

outputLayerWeights.Add(outputLayerWeightsItem);

}

}

}

ПРИЛОЖЕНИЕ И

*(справочное)*

Листинг кода класса CoefficientsCalculator

public class CoefficientsCalculator

{

public static double[] calculateRealCoefficients(double[] prediction)

{

double[] realCoefficients = new double[3];

for (int i = 0; i < 3; i++)

realCoefficients[i] = 1 / prediction[i];

return realCoefficients;

}

public static double[] calculateAdjustedCoefficients(double[] prediction)

{

double[] adjustedCoefficients = new double[3];

Dictionary<string, int> indexes = new Dictionary<string, int>();

double changes = 0.15;

double additionalChanges = 0;

int i = 0;

for (i = 0; i < 3; i++)

{

if (prediction[i] == prediction.Max())

indexes["max"] = i;

else if (prediction[i] == prediction.Min() && !indexes.ContainsKey("min"))

indexes["min"] = i;

else

indexes["medium"] = i;

}

if ((prediction[indexes["min"]] - 0.125) > 0)

{

if ((prediction[indexes["min"]] - 0.125) > 0.07)

additionalChanges = 0.07;

else

additionalChanges = prediction[indexes["min"]] - 0.125;

}

prediction[indexes["min"]] -= additionalChanges;

prediction[indexes["max"]] += changes \* 0.6818 + additionalChanges \* 0.6818;

prediction[indexes["medium"]] += changes \* 0.3182 + additionalChanges \* 0.3182;

for (i = 0; i < 3; i++)

adjustedCoefficients[i] = 1 / prediction[i];

return adjustedCoefficients;

}}

ПРИЛОЖЕНИЕ К

*(справочное)*

Листинг кода класса MatchInfo

public class MatchInfo

{

public int id;

public TeamInfo firstTeam;

public TeamInfo secondTeam;

public int realResult;

public MatchInfo(Match matchInfo, TeamInfo \_firstTeam, TeamInfo \_secondTeam)

{

id = matchInfo.Id;

int.TryParse(matchInfo.Result, out realResult);

firstTeam = \_firstTeam;

secondTeam = \_secondTeam;

}

public double[] getMatchParams()

{

double[] matchParams = new double[18];

List<Parameter> firstTeamParameters = firstTeam.getListOfParameters();

List<Parameter> secondTeamParameters = secondTeam.getListOfParameters();

int i = 0;

foreach (Parameter param in firstTeamParameters)

{

matchParams[i++] = param.value;

}

foreach (Parameter param in secondTeamParameters)

{

matchParams[i++] = param.value;

}

return matchParams;

}

}

# ПРИЛОЖЕНИЕ Л

*(обязательное)*

Спецификация программного дипломного проекта

# ПРИЛОЖЕНИЕ М

*(обязательное)*

Ведомость документов