1. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

**4.1** Квалиметричесикй анализ

**4.1.1** Расчет силы команды

Сила команды является основным и единственным параметром, по которому квалиметрический анализатор принимает решение об исходе события. Поэтому расчет силы команды является основным методом данного анализатора.

У каждого параметра команды есть свой вес и определено, является ли он стимулятором, то есть увеличится ли сила команды при увеличении данного параметра. Однако просто сложить параметры, умноженные на веса, не является лучшим решением, так как результат не будет зависеть от взаимного отношения параметров, что неверно.

Для вычисления силы команды предназначен класс TeamPowerCalculator. Первым этапом квалиметрического анализа является вычисление относительного значения каждого из параметров, присутствующих у команды. Данные операции производятся в методе calculateRelativeParameters, который принимает в себя пару однотипных параметров обеих команд. Рассмотрим фрагмент его кода, приведенный ниже.

В данном коде вначале вычисляется сумма принятых параметров. Если данная сумма равна нулю, то дальнейшие действия бессмысленны и относительные параметры равны нулю. Иначе вычисляется значение каждого параметра относительно суммы и, если он не является стимулятором, то производится его инвертирование относительно единицы.

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;

}

Далее каждый из полученных относительных параметров добавляется к силе команды, которая изначально равна нулю. В конце расчета сила каждой команды нормируется относительно их суммы, что позволяет получить вероятность ее победы от нуля до единицы. Данное действие показано в коде ниже:

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

teamPowers[0] /= teamPowersSum;

teamPowers[1] /= teamPowersSum;

**4.1.2** Вычисление вероятностей исходов матчей

Данный метод предсказания является вспомогательным, поэтому необходимость предсказания вероятностей каждого из исходов не является приоритетной для него.

Сила каждой команды после ее нормировки и будет являться вероятностью победы данной команды. Для поддержания консистентности и однотипности данных, возвращаемых всеми предсказателями, необходимо так же предусмотреть вероятность ничьей.

В данном случае вероятность ничьей принята равной отношению вероятности победы более слабой команды к вероятности победы более сильной команды. Такой выбор вероятностей исходов события позволяет получить не просто предсказание, а вероятности победы каждой команды относительно друг друга. Ниже приведен код предсказания:

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

double[] prediction = new double[] {

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

teamPowers[0],

teamPowers[1]

};

return prediction;

* 1. Нечеткие предсказания

Следующим вспомогательным алгоритмом предсказания является алгоритм нечетких предсказаний, также известный как алгоритм на основе нечетких правил.

Суть данного алгоритма заключается в том, что определенный набор параметров команды сравнивается с наборами параметров всех других команд, имеющихся в системе, и при его совпадении выдается результат данного события. Однако, в связи с тем, что в программе используется довольно обширный набор параметров, для полного совпадения необходима огромная база данный. Учитывая это, алгоритм был немного изменен.

**4.2.1** Генерация параметра команды

Подготовительным этапом алгоритма нечетких предсказаний является генерация строкового параметра каждой команды, содержащего иформацию обо всех других параметрах команды. Рассмотрим код метода analizeTeamInfo, выполняющего данные действия:

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;

В данном методе каждому параметру команды ставится в соответсвие цифра от одного до трех. Выбор цифры происходит в соответствии с границами, определенными для данного конкретного параметра. Все полученные цифры конкатенируются в одну строковую константу, которая и является параметром, анализируемым алгоритмом нечетких предсказаний.

**4.2.2** Вычисление вероятностей исходов матчей

Как было сказано выше, данный алгоритм был изменен в связи с недостаточным размером базы данных игр. Основные изменения заключаются в том, что поиск происходит не до полного совпадения строки-параметра, а проходит по всей базе данных, вычисляя схожесть строк-параметров для каждого из трех возможных исходов матча. Рассмотрим код метода predict, совершающего вычисление данных вероятностей:

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

int i = 0;

string matchFuzzyInfo = "";

Для каждого матча вычисляется параметр для сравнения, получаемый путем склеивания строк-параметров участвующих в нем команд.

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;

Начинается цикл по всем матчам.

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;

В показанном выше цикле матчи классифицируются по их результату. Для каждого результата заведена переменная, показывающая степень подобия данного и предсказываемого матча. С помощью этих переменных в конце цикла получается характеристика подобия исхода анализируемого матча с каждым из возможных исходов матчей.

Для получения вероятности наступления каждого из исходов степень подобия делится на размер строки-параметра матча. Таким образом, изменения, внесенные в алгоритм, позволяют не только не зависеть от размера базы данных, но и улучшают выходные параметры алгоритма. Вместо одного конкретного результата без вероятности его наступления, алгоритм выдает вероятности всех возможных исходов.

**4.3** Предсказание на основе нейронной сети РБФ

**4.3.1** Обучение нейронной сети

Нейронная сеть состоит из нейронов трех слоев:

– входного;

– внутренного;

– выходного.

В сети РБФ нейроны внутреннего слоя представляют собой РБФ-ячейки, устанавливаемые в возбужденное состояние активационной функцией. Данная сеть и ее активационная функция приведены на рисунке 4.1.

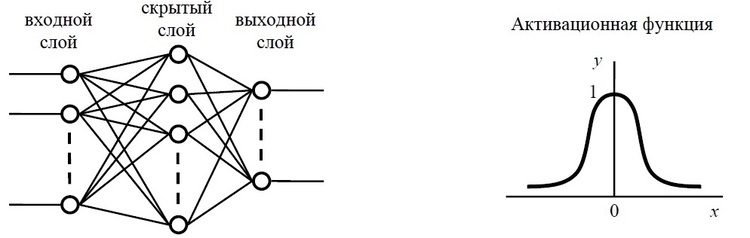


Рисунок 4.1 – Нейронная сеть РБФ и активационная функцияя нейронов внутреннего слоя

Основным этапом в реализации работы нейронной сети является ее обучение. Для этого используется обучающая выборка, от размеров и качества которой напрямую зависит последующая точность работы сети. Рассмотрим алгоритм обучения, представленный методом TeachNeuralNetwork класса RadialBasisFunctionNetwork.

Данный метод итерационно обучает нейронную сеть до тех пор, пока не будет достигнута максимальная величина ошибки, переданная в параметрах. Суть обучения состоит в следующем: обучение РБФ-ячеек происходит за одну итерацию, далее лишь подбираются подходящие веса нейронов выходного слоя. Рассмотрим фрагмент кода, отвечающий за обучения РБФ-ячейки:

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

Инициализация всех ячеек начальными значениями:

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;

}

Если в сети более одной РБФ-ячейки, то для каждой из них вычисляем величину стандартного отклонения. Эта величина представляет собой расстояние между двумя соседними ячейками в многомерном пространстве:

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();

После того, как все нейроны внутреннего слоя обучены, происходит вычисление весов нейронов выходного слоя. За это отвечает фрагмент кода, приведенный ниже:

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];

}

}

Величина изменения веса выходного нейрона за одну итерацию зависит от величины ошибки в данной итерации и от заданной скорости обучения нейронной сети.

**4.3.2** Сохранение состояния обученной нейронной сети

Так как при каждом запуске программы заново обучать нейронную сеть глупо, то было принято решение хранить состояние обученной нейронной сети в базе данных. Для этого необходимо подготовить информацию о нейронах внутренного и выходного слоев к сохранению. Этим занимается метод prepareDataForSaving, код которого приведен ниже:

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);

}

**4.3.3** Классификация матча нейронной сетью

Классификация матча заключается в определении вероятностей каждого из его исходов нейронной сетью. Рассомтрим код метода ClassifyMatch, выполняющего дынные действия:

double[] rbfNeurons = new double[rbfCount];

double[] outputNeurons = new double[outputCount];

Вычисляем выходное значение каждой РБФ-ячейки:

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));

}

Вычисляем итоговое значение нейронов выходного слоя на основании значений нейронов внутреннего слоя и весов нейронов выходного слоя:

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;

}

Возвращаем результаты классификации, которые и являются искомыми вероятностями:

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;

**4.4** Вычисление итоговых вероятностей

На основании трех описанных выше реализаций алгоритмов необходимо предоставить пользователю итоговые вероятности исходов матча.

Так как предсказание нейронной сети является наиболее точным, то его вес выбран максимальным из всех трех и равняется единице. Веса квалиметрического а нечеткого анализатора выбраны 0.3 и 0.33 соответственно. Также предусмотрена возможность того, что реализованный предсказатель допустит ошибку в вычислениях и предсказанная вероятность будет неоправданно малой. Ниже приведен код метода predict класса GeneralPredictor, реализующий вычисление итоговых вероятностей:

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();

Нормализация вероятностей относительно единицы:

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

finalPrediction[i] /= sum;

return finalPrediction;