МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение

высшего профессионального образования

«Ижевский государственный технический университет

имени М.Т. Калашникова»

Кафедра «Программное обеспечение»

Отчет

по лабораторной работе на тему

«Построение дерева решений по алгоритму ID3»

по дисциплине

«Математические основы искусственного интеллекта»

Выполнил:

ст-т гр. Б06-191-2 М. П. Хасанов

Принял преподаватель: А. В. Коробейников

Ижевск 2018

СОДЕРЖАНИЕ

1. Описание задания3

2. Исходный код4

5. Вывод5

1. ОПИСАНИЕ ЗАДАНИЯ

Построить дерево принятий решений с помощью алгоритма ID3 на примере погодных данных о том, был ли сыгран матч.

Исходные данные

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Outlook** | **Temp.** | **Humidity** | **Wind** | **Decision** |
| 1 | Sunny | Hot | High | Weak | No |
| 2 | Sunny | Hot | High | Strong | No |
| 3 | Overcast | Hot | High | Weak | Yes |
| 4 | Rain | Mild | High | Weak | Yes |
| 5 | Rain | Cool | Normal | Weak | Yes |
| 6 | Rain | Cool | Normal | Strong | No |
| 7 | Overcast | Cool | Normal | Strong | Yes |
| 8 | Sunny | Mild | High | Weak | No |
| 9 | Sunny | Cool | Normal | Weak | Yes |
| 10 | Rain | Mild | Normal | Weak | Yes |
| 11 | Sunny | Mild | Normal | Strong | Yes |
| 12 | Overcast | Mild | High | Strong | Yes |
| 13 | Overcast | Hot | Normal | Weak | Yes |
| 14 | Rain | Mild | High | Strong | No |

2. ИСХОДНЫЙ КОД

namespace id3

{

class ID3Builder

{

private const string PositiveDecision = "Yes";

private const string NegativeDecision = "No";

private static readonly Dictionary<Result, string> Decisions = new Dictionary<Result, string>

{

{

Result.Positive,

PositiveDecision

},

{

Result.Negative,

NegativeDecision

},

};

private readonly string \_decisionFactor;

private readonly Dictionary<string, List<string>> \_valuesOfFactors;

private readonly List<Case> \_allCases;

public ID3Builder(List<Case> cases, string decisionFactor)

{

\_decisionFactor = decisionFactor;

\_allCases = cases;

\_valuesOfFactors = GetValuesOfFactors(cases);

}

public static Dictionary<string, List<string>> GetValuesOfFactors(IEnumerable<Case> cases) => cases

.SelectMany(c => c.Factors)

.GroupBy(pair => pair.Key)

.ToDictionary(group => group.Key, group => group.Select(f => f.Value).Distinct().ToList());

public Node Build()

{

var factors = \_valuesOfFactors.Keys.Where(factor => factor != \_decisionFactor).ToList();

var targetFactor = GetBestFactor(\_allCases, factors);

Console.WriteLine($"BestFactor is {targetFactor}");

return Build(\_allCases, factors.Where(factor => factor != targetFactor).ToList(), targetFactor);

}

private Node Build(List<Case> cases, List<string> factors, string targetFactor)

{

var node = new Node {

FactorName = targetFactor,

Childrens = new List<(string factorValue, string factorName, Node child)>(),

Result = Result.None,

};

if (SetSingleRootIfPossible(node, cases))

return node;

var values = \_valuesOfFactors[targetFactor];

foreach (var targetFactorValue in values) {

var casesWithCurrentValue = cases.Where(c => c.Factors[targetFactor] == targetFactorValue).ToList();

var appendingNode = (targetFactorValue, factorName: (string)null, node: new Node());

if (SetSingleRootIfPossible(appendingNode.node, casesWithCurrentValue))

{

Console.WriteLine($"all cases have result {appendingNode.node.Result} for factorValue: {appendingNode.Item1}");

node.Childrens.Add(

appendingNode

);

continue;

}

if (casesWithCurrentValue.Any()) {

var targetFactorForValue = GetBestFactor(casesWithCurrentValue, factors);

var factorsWithoutTarget = factors.Where(f => f != targetFactorForValue && f != targetFactor).ToList();

appendingNode.node = Build(casesWithCurrentValue, factorsWithoutTarget, targetFactorForValue);

appendingNode.factorName = targetFactorForValue;

}

else {

var allCasesWithThisValue = \_allCases.Where(c => c.Factors[targetFactor] == targetFactorValue).ToList();

var successCount = allCasesWithThisValue.Count(c => c.Factors[\_decisionFactor] == Decisions[Result.Positive]);

var result = successCount \* 2 >= allCasesWithThisValue.Count ? Result.Positive : Result.Negative;

appendingNode.node.Result = result; }

Console.WriteLine($"target factor {appendingNode.factorName} for factorValue: {appendingNode.Item1}");

node.Childrens.Add(

appendingNode

); }

return node;

}

private string GetBestFactor(List<Case> cases, IReadOnlyList<string> factors)

{

var bestFactor = factors.ElementWithMaxCost(otherProperty => CalculateGain(cases, otherProperty));

return bestFactor;

}

private double CalculateGain(List<Case> cases, string calculatingProperty)

{

var summedEntropyForProperty = \_valuesOfFactors[calculatingProperty].Sum(v => CalculateGainForPropertyValue(v));

var entropy = CalculateEntropy(cases);

var gain = entropy - summedEntropyForProperty;

Console.WriteLine($"gain for property {calculatingProperty}: {gain:F4}");

return gain;

double CalculateGainForPropertyValue(string calculatingPropertyValue)

{

var casesWithValue = cases.Where(c => c.Factors[calculatingProperty] == calculatingPropertyValue).ToList();

if (!casesWithValue.Any())

return 0;

var pValueOfAllCases = (double)casesWithValue.Count / cases.Count;

return CalculateEntropy(casesWithValue) \* pValueOfAllCases;

}

}

private double CalculateEntropy(List<Case> cases)

{

var positive = cases.Count(c => c.Factors[\_decisionFactor] == PositiveDecision);

var pPositive = (double)positive / cases.Count;

var pNegative = 1 - pPositive;

if (Math.Abs(pPositive) < double.Epsilon || Math.Abs(pNegative) < double.Epsilon)

return 0;

return -pPositive \* Math.Log(pPositive, 2) - pNegative \* Math.Log(pNegative, 2);}

private bool SetSingleRootIfPossible(Node node, List<Case> examples) {

node.Result = CheckIfAll(Result.Positive, examples)

? Result.Positive

: CheckIfAll(Result.Negative, examples)

? Result.Negative

: Result.None;

return node.Result != Result.None; }

private bool CheckIfAll(Result expectedResult, List<Case> cases) =>

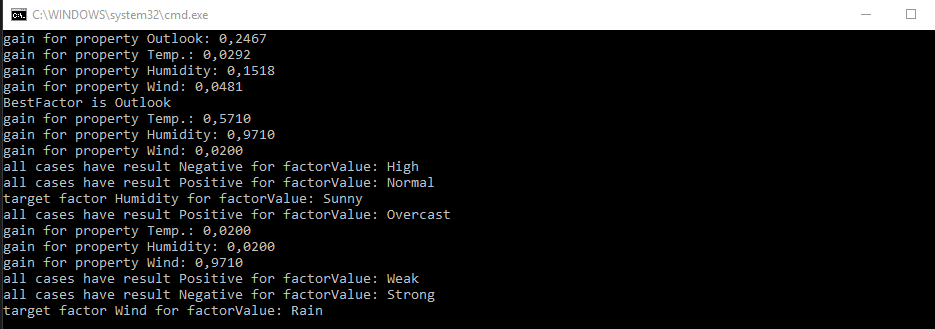
cases.All(@case => @case.Factors[\_decisionFactor] == Decisions[expectedResult]);

} public enum Result {None, Negative, Positive, }

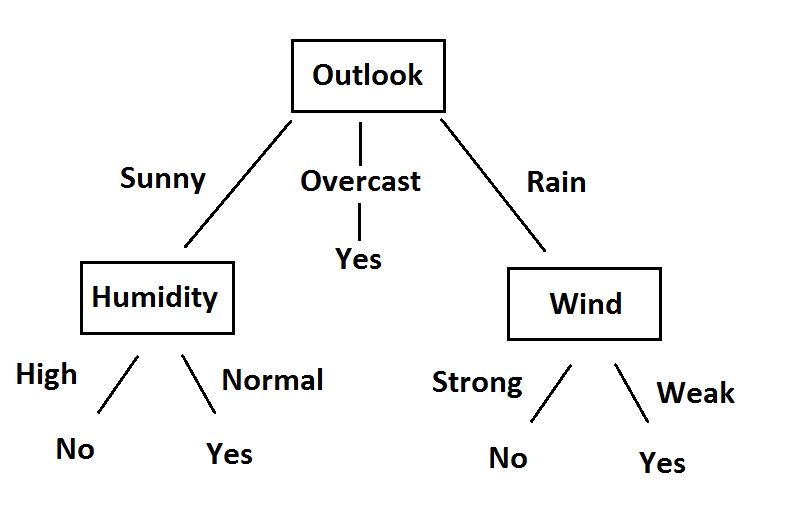
}

3. РЕЗУЛЬТАТ РАБОТЫ

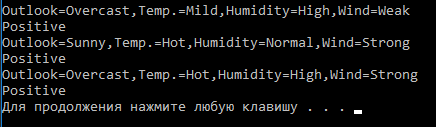
Вычисленные веса для факторов и значений:



Получившееся дерево решений:



Решение принятое программой на основе созданного дерева и новых входных данных:



4. ВЫВОД

В процессе выполнения работы были получены навыки построения решения с помощью алгоритма ID3.