

using System;

using System.Collections.Generic;

using System.Data;

using System.IO;

using System.Linq;

using System.Windows.Forms;

namespace Kod

{

public partial class Form1 : Form

{

private string textVariable;

private bool check = false;

private string str;

private double[] P1;

private string[] Res;

public Form1()

{

InitializeComponent();

resDataGridView.Columns.Add("Column3", "Шеннон-Фано");

resDataGridView.Columns.Add("Column4", "Хаффман");

}

private void button1\_Click(object sender, EventArgs e)

{

if (textVariable != TextBox.Text)

check = false;

if (!check)

str = TextBox.Text.ToLower();

else

str = textVariable.ToLower();

// Расчет вероятностей

Dictionary<char, double> probabilities = CalculateProbabilities(str);

// Шеннон-Фано

var (shannonCodes, lFan) = CalculateShannonFano(probabilities);

// Энтропия

double hFan = CalculateEntropy(probabilities.Values.ToArray());

double dFan = (lFan - hFan) / lFan;

// Хаффман

//Энтропия(H)

//Средняя длина кодов(l)

//Коэффициент избыточности(D)

var (huffmanCodes, lHof) = CalculateHuffman(probabilities);

double dHof = (lHof - hFan) / lHof;

// Обновляем UI

PopulateDataGridView(probabilities, shannonCodes, huffmanCodes);

ShIlabel.Text = "l = " + Math.Round(lFan, 4);

ShHlabel.Text = "H = " + Math.Round(hFan, 4);

ShDlabel.Text = "D = " + Math.Round(dFan, 4);

HafILabel.Text = "l = " + Math.Round(lHof, 4);

HafHlabel.Text = "H = " + Math.Round(hFan, 4);

HafDlabel.Text = "D = " + Math.Round(dHof, 4);

}

private Dictionary<char, double> CalculateProbabilities(string text)

{

Dictionary<char, int> chars = new Dictionary<char, int>();

foreach (char c in text)

{

if (!chars.ContainsKey(c))

chars[c] = 1;

else

chars[c]++;

}

// Считаем вероятности и сортируем их по убыванию

return chars

.ToDictionary(

kvp => kvp.Key,

kvp => (double)kvp.Value / text.Length

)

.OrderByDescending(kvp => kvp.Value) // Сортировка по убыванию

.ToDictionary(kvp => kvp.Key, kvp => kvp.Value);

}

private void PopulateDataGridView(Dictionary<char, double> probabilities, string[] shannonCodes, List<string> huffmanCodes)

{

resDataGridView.Rows.Clear();

int i = 0;

foreach (var pair in probabilities)

{

resDataGridView.Rows.Add();

resDataGridView[0, i].Value = pair.Key;

resDataGridView[1, i].Value = Math.Round(pair.Value, 4);

resDataGridView[2, i].Value = shannonCodes[i];

resDataGridView[3, i].Value = huffmanCodes[i];

i++;

}

}

private double CalculateEntropy(double[] probabilities)

{

return probabilities.Sum(p => f(p));

}

private (string[], double) CalculateShannonFano(Dictionary<char, double> probabilities)

{

ShannonFanoEncoder shannonFanoEncoder = new ShannonFanoEncoder(probabilities.Values.ToArray());

string[] codes = shannonFanoEncoder.Encode();

double avgLength = probabilities.Values

.Zip(codes, (prob, code) => prob \* code.Length)

.Sum();

return (codes, avgLength);

}

private (List<string>, double) CalculateHuffman(Dictionary<char, double> probabilities)

{

HuffmanTree huffmanTree = new HuffmanTree();

huffmanTree.Build(probabilities);

List<string> codes = huffmanTree.ReturnAlphabet();

double avgLength = probabilities.Values

.Zip(codes, (prob, code) => prob \* code.Length)

.Sum();

return (codes, avgLength);

}

private double f(double x)

{

return -x \* Math.Log(x, 2);

}

private void button2\_Click(object sender, EventArgs e)

{

check = true;

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Текстовые файлы (\*.txt)|\*.txt|Все файлы (\*.\*)|\*.\*";

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

string filePath = openFileDialog.FileName;

string fileText = File.ReadAllText(filePath);

textVariable = fileText;

TextBox.Text = textVariable;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Kod

{

internal class HuffmanNode

{

public int Frequency { get; set; }

public char Character { get; set; }

public HuffmanNode Left { get; set; }

public HuffmanNode Right { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Kod

{

internal class HuffmanTree

{

private List<Node> nodes = new List<Node>();

public Node Root { get; set; }

public Dictionary<char, double> Frequencies = new Dictionary<char, double>();

public void Build(Dictionary<char, double> organ\_harvest)

{

Frequencies = organ\_harvest;

foreach (KeyValuePair<char, double> symbol in Frequencies)

{

nodes.Add(new Node() { Symbol = symbol.Key, Frequency = symbol.Value });

}

while (nodes.Count > 1)

{

List<Node> orderedNodes = nodes.OrderBy(node => node.Frequency).ToList<Node>();

if (orderedNodes.Count >= 2)

{

List<Node> taken = orderedNodes.Take(2).ToList<Node>();

Node parent = new Node()

{

Symbol = '\*',

Frequency = taken[0].Frequency + taken[1].Frequency,

Left = taken[1],

Right = taken[0]

};

nodes.Remove(taken[0]);

nodes.Remove(taken[1]);

nodes.Add(parent);

}

this.Root = nodes.FirstOrDefault();

}

}

public List<string> ReturnAlphabet()

{

List<string> tmp = new List<string>();

foreach (KeyValuePair<char, double> symbol in Frequencies)

{

List<bool> encodedSymbol = this.Root.Traverse(symbol.Key, new List<bool>());

tmp.Add(new string(encodedSymbol.Select(x => x ? '0' : '1').ToArray()));

}

return tmp;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Kod

{

internal class HuffmanTree

{

private List<Node> nodes = new List<Node>();

public Node Root { get; set; }

public Dictionary<char, double> Frequencies = new Dictionary<char, double>();

public void Build(Dictionary<char, double> organ\_harvest)

{

Frequencies = organ\_harvest;

foreach (KeyValuePair<char, double> symbol in Frequencies)

{

nodes.Add(new Node() { Symbol = symbol.Key, Frequency = symbol.Value });

}

while (nodes.Count > 1)

{

List<Node> orderedNodes = nodes.OrderBy(node => node.Frequency).ToList<Node>();

if (orderedNodes.Count >= 2)

{

List<Node> taken = orderedNodes.Take(2).ToList<Node>();

Node parent = new Node()

{

Symbol = '\*',

Frequency = taken[0].Frequency + taken[1].Frequency,

Left = taken[1],

Right = taken[0]

};

nodes.Remove(taken[0]);

nodes.Remove(taken[1]);

nodes.Add(parent);

}

this.Root = nodes.FirstOrDefault();

}

}

public List<string> ReturnAlphabet()

{

List<string> tmp = new List<string>();

foreach (KeyValuePair<char, double> symbol in Frequencies)

{

List<bool> encodedSymbol = this.Root.Traverse(symbol.Key, new List<bool>());

tmp.Add(new string(encodedSymbol.Select(x => x ? '0' : '1').ToArray()));

}

return tmp;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

namespace Kod

{

public class ShannonFanoEncoder

{

private double[] probabilities;

private string[] codes;

public ShannonFanoEncoder(double[] probabilities)

{

this.probabilities = probabilities;

this.codes = new string[probabilities.Length];

}

public string[] Encode()

{

Fano(0, probabilities.Length - 1);

return codes;

}

private void Fano(int left, int right)

{

if (left < right)

{

int middle = FindSplitPoint(left, right);

for (int i = left; i <= right; i++)

{

codes[i] = codes[i] ?? ""; // Инициализация строки

codes[i] += (i <= middle) ? "1" : "0";

}

Fano(left, middle);

Fano(middle + 1, right);

}

}

private int FindSplitPoint(int left, int right)

{

double leftSum = probabilities[left];

double rightSum = probabilities.Skip(left + 1).Take(right - left).Sum();

int splitIndex = left;

double minDifference = Math.Abs(leftSum - rightSum);

for (int i = left + 1; i <= right; i++)

{

leftSum += probabilities[i];

rightSum -= probabilities[i];

double currentDifference = Math.Abs(leftSum - rightSum);

if (currentDifference < minDifference)

{

minDifference = currentDifference;

splitIndex = i;

}

}

return splitIndex;

}

}

}