БУ ВО «Сургутский государственный университет»

Политехнический институт

Кафедра автоматизированных систем обработки информации и управления

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

ПО ДИСЦИПЛИНЕ «Теория Информации»

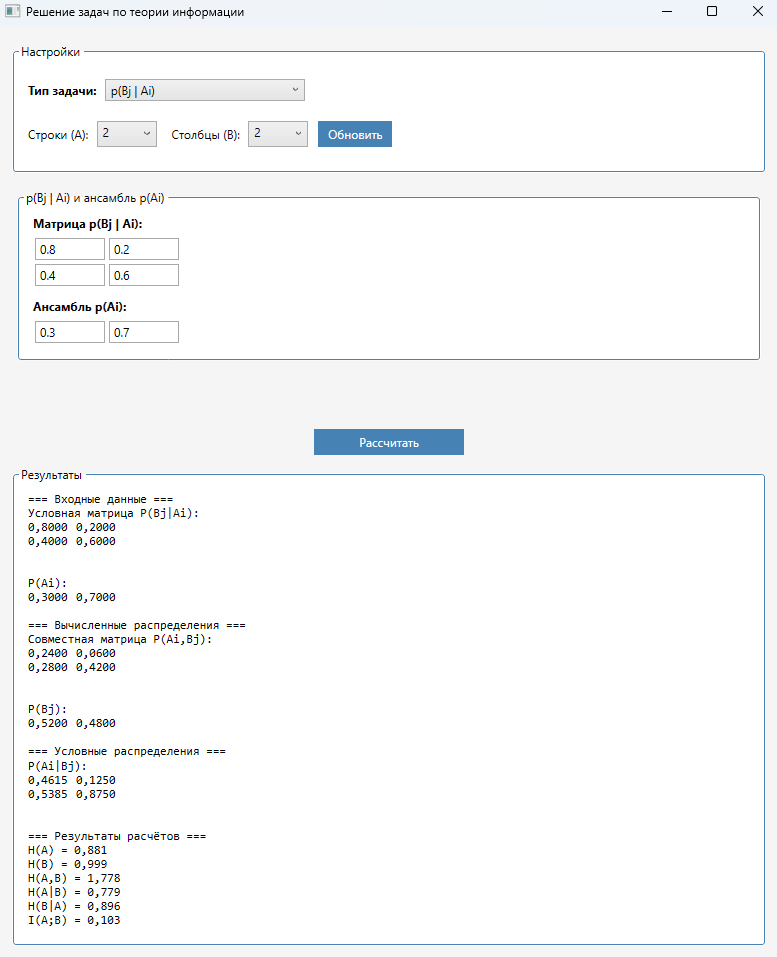
Выполнил: студент группы №606-12,

Речук Дмитрий Максимович

Принял: ст. преподаватель кафедры АСОИУ,

Гавриленко Анна Владимировна

Сургут 2025



using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace lab1

{

public partial class MainWindow : Window, INotifyPropertyChanged

{

private bool \_isJointVisible = true;

private bool \_isCondWgivenZVisible = false;

private bool \_isCondZgivenWVisible = false;

public bool IsJointVisible

{

get => \_isJointVisible;

set { \_isJointVisible = value; OnPropertyChanged(nameof(IsJointVisible)); }

}

public bool IsCondWgivenZVisible

{

get => \_isCondWgivenZVisible;

set { \_isCondWgivenZVisible = value; OnPropertyChanged(nameof(IsCondWgivenZVisible)); }

}

public bool IsCondZgivenWVisible

{

get => \_isCondZgivenWVisible;

set { \_isCondZgivenWVisible = value; OnPropertyChanged(nameof(IsCondZgivenWVisible)); }

}

private List<List<ValueWrapper>> matrixJoint = new();

private List<List<ValueWrapper>> matrixCondWgivenZ = new();

private List<List<ValueWrapper>> matrixCondZgivenW = new();

private List<ValueWrapper> ensembleZ = new();

private List<ValueWrapper> ensembleW = new();

public MainWindow()

{

InitializeComponent();

DataContext = this;

InitializeInputs(2, 4);

}

private void TaskTypeComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

UpdateVisibility(TaskTypeComboBox.SelectedIndex);

}

private void UpdateVisibility(int selectedIndex)

{

IsJointVisible = (selectedIndex == 0);

IsCondWgivenZVisible = (selectedIndex == 1);

IsCondZgivenWVisible = (selectedIndex == 2);

}

private void UpdateMatrixButton\_Click(object sender, RoutedEventArgs e)

{

int rows = RowsComboBox.SelectedIndex + 1;

int cols = ColsComboBox.SelectedIndex + 1;

InitializeInputs(rows, cols);

}

private void InitializeInputs(int rows, int cols)

{

matrixJoint = CreateMatrix(rows, cols);

matrixCondWgivenZ = CreateMatrix(rows, cols);

matrixCondZgivenW = CreateMatrix(rows, cols);

ensembleZ = CreateVector(rows);

ensembleW = CreateVector(cols);

MatrixJointInput.ItemsSource = matrixJoint;

MatrixCondWgivenZInput.ItemsSource = matrixCondWgivenZ;

MatrixCondZgivenWInput.ItemsSource = matrixCondZgivenW;

EnsembleZInput.ItemsSource = ensembleZ;

EnsembleWInput.ItemsSource = ensembleW;

}

private List<List<ValueWrapper>> CreateMatrix(int rows, int cols)

{

var matrix = new List<List<ValueWrapper>>();

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

{

var row = new List<ValueWrapper>();

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

{

row.Add(new ValueWrapper { Value = 0.0 });

}

matrix.Add(row);

}

return matrix;

}

private List<ValueWrapper> CreateVector(int length)

{

var vector = new List<ValueWrapper>();

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

{

vector.Add(new ValueWrapper { Value = 0.0 });

}

return vector;

}

private void CalculateButton\_Click(object sender, RoutedEventArgs e)

{

int selectedTask = TaskTypeComboBox.SelectedIndex;

string result = "";

try

{

if (selectedTask == 0) // p(Ai, Bj)

{

var matrix = GetMatrix(matrixJoint);

result = SolveFromJoint(matrix);

}

else if (selectedTask == 1) // p(Bj | Ai) и p(Ai)

{

var condMatrix = GetMatrix(matrixCondWgivenZ);

var ensemble = GetVector(ensembleZ);

result = SolveFromCondWgivenZ(condMatrix, ensemble);

}

else if (selectedTask == 2) // p(Ai | Bj) и p(Bj)

{

var condMatrix = GetMatrix(matrixCondZgivenW);

var ensemble = GetVector(ensembleW);

result = SolveFromCondZgivenW(condMatrix, ensemble);

}

}

catch (Exception ex)

{

result = "Ошибка расчёта: " + ex.Message;

}

ResultsTextBlock.Text = result;

}

private double[,] GetMatrix(List<List<ValueWrapper>> input)

{

int rows = input.Count;

int cols = input[0].Count;

var matrix = new double[rows, cols];

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

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

matrix[i, j] = input[i][j].Value;

return matrix;

}

private double[] GetVector(List<ValueWrapper> input)

{

return input.Select(x => x.Value).ToArray();

}

private string SolveFromJoint(double[,] pJoint)

{

int rows = pJoint.GetLength(0);

int cols = pJoint.GetLength(1);

double[] pZ = new double[rows];

double[] pW = new double[cols];

// Вычисляем маргинальные распределения

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

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

pZ[i] += pJoint[i, j];

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

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

pW[j] += pJoint[i, j];

// Вычисляем условные матрицы

double[,] pWgivenZ = new double[rows, cols];

double[,] pZgivenW = new double[rows, cols];

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

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

{

pWgivenZ[i, j] = pZ[i] > 0 ? pJoint[i, j] / pZ[i] : 0;

pZgivenW[i, j] = pW[j] > 0 ? pJoint[i, j] / pW[j] : 0;

}

// Вычисляем энтропии

double Hz = -pZ.Where(p => p > 0).Sum(p => p \* Math.Log(p, 2));

double Hw = -pW.Where(p => p > 0).Sum(p => p \* Math.Log(p, 2));

double HzW = -pJoint.Cast<double>().Where(p => p > 0).Sum(p => p \* Math.Log(p, 2));

double HzGivenW = HzW - Hw;

double HwGivenZ = HzW - Hz;

double IZW = Hz + Hw - HzW;

// Формируем вывод всех матриц

string output = "=== Входные данные ===\n";

output += "Совместная матрица P(Ai,Bj):\n" + MatrixToString(pJoint) + "\n\n";

output += "=== Вычисленные распределения ===\n";

output += "P(Ai):\n" + VectorToString(pZ) + "\n\n";

output += "P(Bj):\n" + VectorToString(pW) + "\n\n";

output += "=== Условные распределения ===\n";

output += "P(Bj|Ai):\n" + MatrixToString(pWgivenZ) + "\n\n";

output += "P(Ai|Bj):\n" + MatrixToString(pZgivenW) + "\n\n";

output += "=== Результаты расчётов ===\n";

output += FormatResult(Hz, Hw, HzW, HzGivenW, HwGivenZ, IZW);

return output;

}

private string SolveFromCondWgivenZ(double[,] pWgivenZ, double[] pZ)

{

int rows = pWgivenZ.GetLength(0);

int cols = pWgivenZ.GetLength(1);

// Вычисляем совместную матрицу и P(Bj)

double[,] pJoint = new double[rows, cols];

double[] pW = new double[cols];

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

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

pJoint[i, j] = pZ[i] \* pWgivenZ[i, j];

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

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

pW[j] += pJoint[i, j];

// Вычисляем P(Ai|Bj)

double[,] pZgivenW = new double[rows, cols];

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

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

pZgivenW[i, j] = pW[j] > 0 ? pJoint[i, j] / pW[j] : 0;

// Вычисляем энтропии

double Hz = -pZ.Where(p => p > 0).Sum(p => p \* Math.Log(p, 2));

double Hw = -pW.Where(p => p > 0).Sum(p => p \* Math.Log(p, 2));

double HzW = -pJoint.Cast<double>().Where(p => p > 0).Sum(p => p \* Math.Log(p, 2));

double HzGivenW = HzW - Hw;

double HwGivenZ = HzW - Hz;

double IZW = Hz + Hw - HzW;

// Формируем вывод всех матриц

string output = "=== Входные данные ===\n";

output += "Условная матрица P(Bj|Ai):\n" + MatrixToString(pWgivenZ) + "\n\n";

output += "P(Ai):\n" + VectorToString(pZ) + "\n\n";

output += "=== Вычисленные распределения ===\n";

output += "Совместная матрица P(Ai,Bj):\n" + MatrixToString(pJoint) + "\n\n";

output += "P(Bj):\n" + VectorToString(pW) + "\n\n";

output += "=== Условные распределения ===\n";

output += "P(Ai|Bj):\n" + MatrixToString(pZgivenW) + "\n\n";

output += "=== Результаты расчётов ===\n";

output += FormatResult(Hz, Hw, HzW, HzGivenW, HwGivenZ, IZW);

return output;

}

private string SolveFromCondZgivenW(double[,] pZgivenW, double[] pW)

{

int rows = pZgivenW.GetLength(0);

int cols = pZgivenW.GetLength(1);

// Вычисляем совместную матрицу и P(Ai)

double[,] pJoint = new double[rows, cols];

double[] pZ = new double[rows];

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

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

pJoint[i, j] = pW[j] \* pZgivenW[i, j];

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

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

pZ[i] += pJoint[i, j];

// Вычисляем P(Bj|Ai)

double[,] pWgivenZ = new double[rows, cols];

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

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

pWgivenZ[i, j] = pZ[i] > 0 ? pJoint[i, j] / pZ[i] : 0;

// Вычисляем энтропии

double Hz = -pZ.Where(p => p > 0).Sum(p => p \* Math.Log(p, 2));

double Hw = -pW.Where(p => p > 0).Sum(p => p \* Math.Log(p, 2));

double HzW = -pJoint.Cast<double>().Where(p => p > 0).Sum(p => p \* Math.Log(p, 2));

double HzGivenW = HzW - Hw;

double HwGivenZ = HzW - Hz;

double IZW = Hz + Hw - HzW;

// Формируем вывод всех матриц

string output = "=== Входные данные ===\n";

output += "Условная матрица P(Ai|Bj):\n" + MatrixToString(pZgivenW) + "\n\n";

output += "P(Bj):\n" + VectorToString(pW) + "\n\n";

output += "=== Вычисленные распределения ===\n";

output += "Совместная матрица P(Ai,Bj):\n" + MatrixToString(pJoint) + "\n\n";

output += "P(Ai):\n" + VectorToString(pZ) + "\n\n";

output += "=== Условные распределения ===\n";

output += "P(Bj|Ai):\n" + MatrixToString(pWgivenZ) + "\n\n";

output += "=== Результаты расчётов ===\n";

output += FormatResult(Hz, Hw, HzW, HzGivenW, HwGivenZ, IZW);

return output;

}

// Вспомогательные методы для преобразования в строки (оставить без изменений)

private string MatrixToString(double[,] matrix)

{

int rows = matrix.GetLength(0);

int cols = matrix.GetLength(1);

string result = "";

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

{

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

{

result += $"{matrix[i, j]:F4}\t";

}

result += "\n";

}

return result;

}

private string VectorToString(double[] vector)

{

string result = "";

foreach (var value in vector)

{

result += $"{value:F4}\t";

}

return result;

}

private string FormatResult(double Hz, double Hw, double HzW, double HzGivenW, double HwGivenZ, double IZW)

{

return $"H(A) = {Hz:F3} \n" +

$"H(B) = {Hw:F3} \n" +

$"H(A,B) = {HzW:F3} \n" +

$"H(A|B) = {HzGivenW:F3} \n" +

$"H(B|A) = {HwGivenZ:F3} \n" +

$"I(A;B) = {IZW:F3} ";

}

public event PropertyChangedEventHandler PropertyChanged;

private void OnPropertyChanged(string propertyName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

public class ValueWrapper

{

public double Value { get; set; }

}

}