Безпалий Марко Леонідович
КН-М922в

Лабораторна робота №1

ПОПЕРЕДНЯ ОБРОБКА ЕКСПЕРИМЕНТАЛЬНИХ ДАНИХ

Мета роботи: вивчення принципів побудови таблиці експериментальних даних (ТЕД) типу «об'єкт-ознака» та методів їх попередньої обробки.

Варіант 1

Завдання:

1. Сформувати ТЕД із файлу з початковими даними
2. Обчислити матрицю зав'язків, використовуючи:
   - Коефіцієнт кореляції Пірсона
   - Коефіцієнт рангової кореляції Кендала
3. Виконати центрування та нормування ознак на «одиничний куб»
4. Обчислити матрицю близькості (віддаленості) об'єктів у ТЕД, використовуючи відстань Чебишова

Хід роботи:

Для виконання лабораторної роботи використовувалась мова програмування Java версії 11.

1. Для того, щоб зчитати данні з файлу було створено клас Indication (ознака), який є переліком всіх ознак.

Основна модель у цій роботі – це структура ключ-значення, де ключем виступає ознака, а значенням – список всіх даних для цієї ознаки, тобто стовпець:

```
Map<Indication, List<Double>> indications
```

Маємо наступні ознаки: Leukocytes, Lymphocytes, T-Lymphocytes, T-Helpers, RE-T-Suppressors, Sens. Theophylline, Res. Theophylline, B-Lymphocytes.

2. Обчислення матриць зв'язків виконується для верхньої половини матриці та відображається відносно головній діагоналі, тому що є симетричною.

Алгоритм **побудови** матриці:

```
  public static <T> List<List<Double>> buildSymmetricRelations(Map<T,
List<Double>> data,

BiFunction<List<Double>, List<Double>, Double> mapper) {

        List<T> uncheckedObjectKeys = new ArrayList<>(data.keySet());
        List<List<Double>> result = new
ArrayList<>(uncheckedObjectKeys.size());
```

```
        do {
            T objectKey = uncheckedObjectKeys.get(0);
            List<Double> x1Values = data.get(objectKey);

            List<Double> row = uncheckedObjectKeys.stream()
                    .map(data::get)
                    .map(x2Values -> mapper.apply(x1Values, x2Values))
                    .collect(Collectors.toList());

            uncheckedObjectKeys.remove(objectKey);
            result.add(row);

        } while (CollectionUtils.isNotEmpty(uncheckedObjectKeys));

        return reflectAlongMainDiagonal(result);
    }
```

Другим аргументом методу (із назвою *mapper*) є функція, що приймає два вектори та повертає число, це й будуть наші методи, що вираховують коефіцієнти кореляції.

Метод для знаходження коефіцієнта **Пірсона**:

```
  public static double pearson(List<Double> x1Vector, List<Double>
x2Vector) {
        int size = x1Vector.size();

        double xSum = 0;
        double ySum = 0;
        double xSquareSum = 0;
        double ySquareSum = 0;
        double sumXY = 0;

        for (int i = 0; i < x1Vector.size(); i++) {
            Double x = x1Vector.get(i);
            Double y = x2Vector.get(i);
            xSum += x;
            ySum += y;
            xSquareSum += x * x;
            ySquareSum += y * y;
            sumXY += x * y;
        }

        double denominator = ((size * xSquareSum) - Math.pow(xSum, 2))
* ((size * ySquareSum) - Math.pow(ySum, 2));

        return ((size * sumXY) - (xSum * ySum)) /
Math.sqrt(denominator);
    }
```

Для обчислення коефіцієнту Пірсона вираховуються суми векторів (`xSum, ySum`), суми квадратів векторів (`xSquareSum, ySquareSum`) та сума перемноження їх значень із однаковими індексами (`sumXY`). Всі ці змінні використовуються в кінцевій формулі:

$$\frac{((size * sumXY) - (xSum * ySum))}{\sqrt{((size * xSquareSum) - xSum^2) * ((size * ySquareSum) - ySum^2)}}$$

Метод для знаходження рангового коефіцієнта **Кендала**:

```
public static double kendall(List<Double> x1Vector, List<Double>
x2Vector) {
        List<Map.Entry<Double, Double>> x1ToX2Entries = new
ArrayList<>();

        for (int i = 0; i < x1Vector.size(); i++) {
            AbstractMap.SimpleImmutableEntry<Double, Double> x1ToX2 =
new AbstractMap.SimpleImmutableEntry<>(x1Vector.get(i),
x2Vector.get(i));
            x1ToX2Entries.add(x1ToX2);
        }

        List<Double> x2Sorted = x1ToX2Entries.stream()
                .sorted(Map.Entry.comparingByKey())
                .map(Map.Entry::getValue)
                .collect(Collectors.toList());

        List<Integer> concordant = new ArrayList<>();
        List<Integer> discordant = new ArrayList<>();

        for (int i = 0; i < x2Sorted.size(); i++) {
            Double testValue = x2Sorted.get(i);

            int concordantCount = (int) x2Sorted.stream()
                    .skip(i)
                    .filter(valueBelow -> testValue < valueBelow)
                    .count();

            int discordantCount = (int) x2Sorted.stream()
                    .skip(i)
                    .filter(valueBelow -> testValue > valueBelow)
                    .count();

            concordant.add(concordantCount);
            discordant.add(discordantCount);
        }

        double c = concordant.stream()
                .mapToDouble(Double::valueOf)
```

```
                .sum();

        double d = discordant.stream()
                .mapToDouble(Double::valueOf)
                .sum();

        return (c - d) / (c + d);
    }
```

Спочатку створюється зв'язок x1 до x2 значень, щоб відсортувати їх за ключем. Далі для кожного значення вираховується кількість значень, які стоять нижче, що менше за поточне (`discordant`) та більше за поточне (`concordant`). Знаходимо суму всіх значень окремо для `concordant` та `discordant`, щоб використати їх у кінцевій формулі:

$$\frac{cSum - dSum}{cSum + dSum}$$

Але для того, щоб побудувати матрицю зв'язків із рангових коефіцієнтів Кендала, нам потрібно спершу **вирахувати ранги**. По суті ранги є середнім значенням індексу у відсортованому масиві даних. Тож щоб їх вирахувати нам потрібно відсортувати значення від найбільшого до найменшого та знайти індекс кожного числа в цьому відсортованому списку. Якщо число зустрічається декілька разів, ми беремо середнє значення індексу (останній індекс + перший індекс поділити навпіл). У коді все виглядає наступним чином:

```
  public static Map<Indication, List<Double>>
calculateRanks(Map<Indication, List<Double>> indicationsMap) {
        Map<Indication, List<Double>> ranksMatrix = new
EnumMap<>(Indication.class);

        indicationsMap.forEach((indication, forwardList) -> {
            List<Double> reverseList = forwardList.stream()
                    .sorted(Comparator.reverseOrder())
                    .collect(Collectors.toList());

            List<Double> rankColumn = new ArrayList<>();
            Map<Double, Double> checkedValues = new HashMap<>();

            forwardList.forEach(value -> {
                Double calculatedPreviously =
checkedValues.get(value);

                if (calculatedPreviously == null) {
                    double rank = getRank(value, reverseList);
                    rankColumn.add(rank);
                    checkedValues.put(value, rank);
                } else {
                    rankColumn.add(calculatedPreviously);
                }
```

```
            });

            ranksMatrix.put(indication, rankColumn);
        });

        return ranksMatrix;
    }


    private static double getRank(double x, List<Double> column) {
        int firstIndex = column.indexOf(x) + 1;
        long count = column.stream().filter(value -> value ==
x).count();

        if (count == 1) {
            return firstIndex;
        }

        int lastIndex = column.lastIndexOf(x) + 1;

        return (lastIndex + firstIndex) / 2.0;
    }
```

Для тестування гіпотез (важливості) кожного значення використовувались розподіл Ст’юдента та нормальний розподіл.

```
public int normal(double value) {
        try {
            double zCritical = (Math.abs(value) * Math.sqrt(9.0 *
sizeN * (sizeN - 1.0)))
                    / (Math.sqrt(2.0 * (2.0 * sizeN + 5.0)));

            double zProbability = zCritical > 0
                    ? 1.0 - normalDistribution.cumulativeProbability(-
zCritical, zCritical)
                    : 1.0 -
normalDistribution.cumulativeProbability(zCritical, -zCritical);

            return zProbability < (1.0 - probability)
                    ? 1
                    : 0;
        } catch (MathException e) {
            throw new IllegalStateException(e);
        }
    }

    public int student(double value) {
        try {
            double tCritical = (Math.abs(value) *
Math.sqrt(degreesOfFreedom)) / (Math.sqrt(1 - Math.pow(value, 2)));

            double studentCritical =
studentDistribution.inverseCumulativeProbability(probability);
```

```
            return tCritical < studentCritical
                    ? 0
                    : 1;
        } catch (MathException e) {
            throw new IllegalStateException(e);
        }
    }
```

Було використано бібліотеку Apache Math для знаходження значень розподілів. Для отримання матриці, що містить значення -1, 0, 1, що є зворотнім зв'язком, відсутністю зв'язку та прямим зв'язком відповідно, кожне значення коефіцієнту перетворюється за формулою:

$$\frac{value}{|value|} * significance$$

де significance – результат функції розподілу (`normal` чи `student`).

```
    public List<List<Integer>>
testSignificanceByDistribution(List<List<Double>> matrix,
DoubleToIntFunction distributionCriteria) {

        List<List<Integer>> result = new ArrayList<>(matrix.size());


        for (List<Double> row : matrix) {

            List<Integer> testedRow = new ArrayList<>(row.size());


            for (Double value : row) {

                int significance =
distributionCriteria.applyAsInt(value);

                testedRow.add((int) (value / Math.abs(value)) *
significance);

            }

            result.add(testedRow);

        }

        return result;

    }
```

3. Для центрування та нормування ТЕД за «одиничним кубом» нам потрібно буде знайти **середнє значення** для ознаки та **радіус**.

```
  public static List<Double> unitCube(List<Double> indicationValues) {
        double sum = indicationValues.stream()
                .mapToDouble(Double::doubleValue)
                .sum();

        double center = sum / indicationValues.size();

        double radius = indicationValues.stream()
```

```
                .map(value -> Math.abs(value - center))
                .mapToDouble(Double::doubleValue)
                .max()
                .orElseThrow(() -> new IllegalStateException("Argument
list is empty"));

        return indicationValues.stream()
                .map(value -> (value - center) / radius)
                .collect(Collectors.toList());
    }
```

Середнє значення та радіус використовуються в кінцевій формулі:

$$\frac{value - center}{radius}$$

4. Щоб вирахувати матрицю відстані, перш за все треба змінити логічний напрямок. Якщо раніше це були стовпці, тепер це повинні бути рядки.

Наступний метод змінює логічне направлення та привласнює кожному об’єкту порядковий номер.

```
  private Map<Integer, List<Double>>
splitIndicationsByObject(List<List<Double>> matrixColumns) {

        int objectsCount = matrixColumns.get(0).size();

        Map<Integer, List<Double>> result = new
HashMap<>(objectsCount);

        for (int i = 0; i < objectsCount; i++) {
            ArrayList<Double> row = new ArrayList<>();

            for (List<Double> column : matrixColumns) {
                row.add(column.get(i));
            }

            result.put(i, row);
        }

        return result;
    }
```

Відстань **Чебишева** вираховується наступним методом:

```
  public static double chebyshev(List<Double> x1Vector, List<Double>
x2Vector) {
        if (x1Vector.size() != x2Vector.size()) {
            throw new IllegalArgumentException("Vectors must be with
the same size");
        }

        List<Double> subs = new ArrayList<>();
```

```
        for (int i = 0; i < x1Vector.size(); i++) {
            subs.add(Math.abs(x1Vector.get(i) - x2Vector.get(i)));
        }

        return subs.stream()
                .mapToDouble(Double::doubleValue)
                .max()
                .orElseThrow(() -> new IllegalStateException("Values
are not present"));
    }
```

Логіка методу Чебишева в тому, щоб знайти модуль значення максимальної різниці між двома об’єктами, порівнюючи значення їх ознак.

Щоб побудувати саму матрицю, використовується метод `buildSymmetricRelations`, щоб був описаний вище. Йому на вхід передається матриця, де логічне направлення вже по рядках (результат методу `splitIndicationsByObject`), а функція перетворення – метод Чебишева.

Результати роботи програми

Вибрана вірогідність для розподілів: 0.95

Довжина вибірки: 137

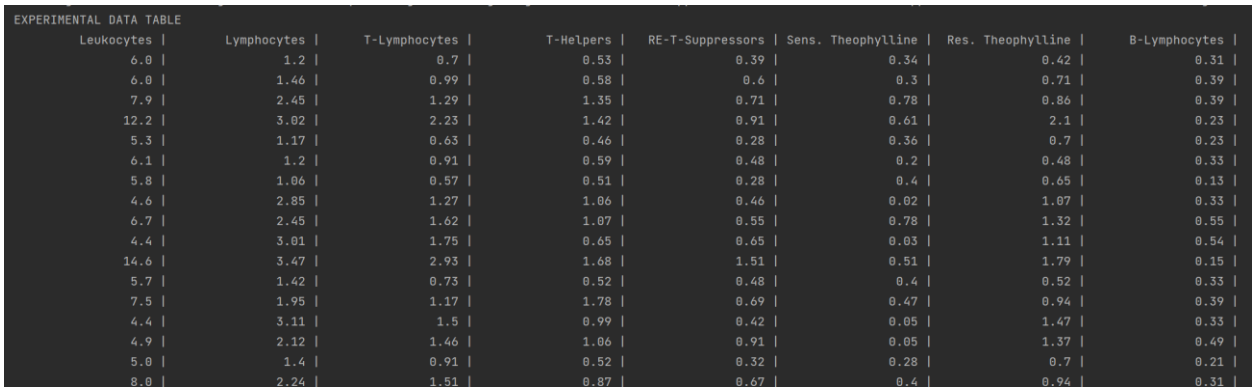
Алгоритм використання методів:

```
final double p = 0.95;

        Map<Indication, List<Double>> indications =
IndicationsFileReader.readFromFile("/lab1_v2.txt");
        int sizeN =
indications.get(Indication.LEUKOCYTES_NUMBER).size();

        DistributionCriteria distributionCriteria =
DistributionCriteria.builder()
                .sizeN(sizeN)
                .probability(p)
                .build();

        ConsoleWriter.printEDT(indications);

        //-------------------------------------------------------------
-----------------------------------------------
        // PEARSON
        //-------------------------------------------------------------
-----------------------------------------------

        List<List<Double>> pearsonRelationMatrix =
Matrices.buildSymmetricRelations(indications, Coefficients::pearson);
```

```
        List<List<Integer>> testedPearson =
Matrices.testSignificanceByDistribution(pearsonRelationMatrix,
distributionCriteria::student);

        ConsoleWriter.printRelationMatrix("pearson",
pearsonRelationMatrix);
        ConsoleWriter.printTestedRelationMatrix("pearson", sizeN, p,
testedPearson);

        //-------------------------------------------------------------
-------------------------------------------------
        // KENDALL
        //-------------------------------------------------------------
-------------------------------------------------

        Map<Indication, List<Double>> indicationRanksMap =
Matrices.calculateRanks(indications);
        List<List<Double>> kendallRelationMatrix =
Matrices.buildSymmetricRelations(indicationRanksMap,
Coefficients::kendall);
        List<List<Integer>> testedKendall =
Matrices.testSignificanceByDistribution(kendallRelationMatrix,
distributionCriteria::normal);

        ConsoleWriter.printRelationMatrix("kendall",
kendallRelationMatrix);
        ConsoleWriter.printTestedRelationMatrix("kendall", sizeN, p,
testedKendall);

        //-------------------------------------------------------------
-------------------------------------------------
        // CENTERING AND NORMALIZATION
        //-------------------------------------------------------------
-------------------------------------------------

        Map<Indication, List<Double>> normalizedIndications =
indications.entrySet().stream()
                .collect(Collectors.toMap(Map.Entry::getKey, entry ->
Normalizations.unitCube(entry.getValue())));

        ConsoleWriter.printNormalizedEDT(normalizedIndications, "UNIT
CUBE");

        //-------------------------------------------------------------
-------------------------------------------------
        // DISTANCE MATRIX
        //-------------------------------------------------------------
-------------------------------------------------

        List<List<Double>> distanceMatrix =
Matrices.getDistanceMatrix(normalizedIndications,
Distances::chebyshev);
```

```
        ConsoleWriter.printDistanceMatrix(distanceMatrix,
"Chebyshev");
```

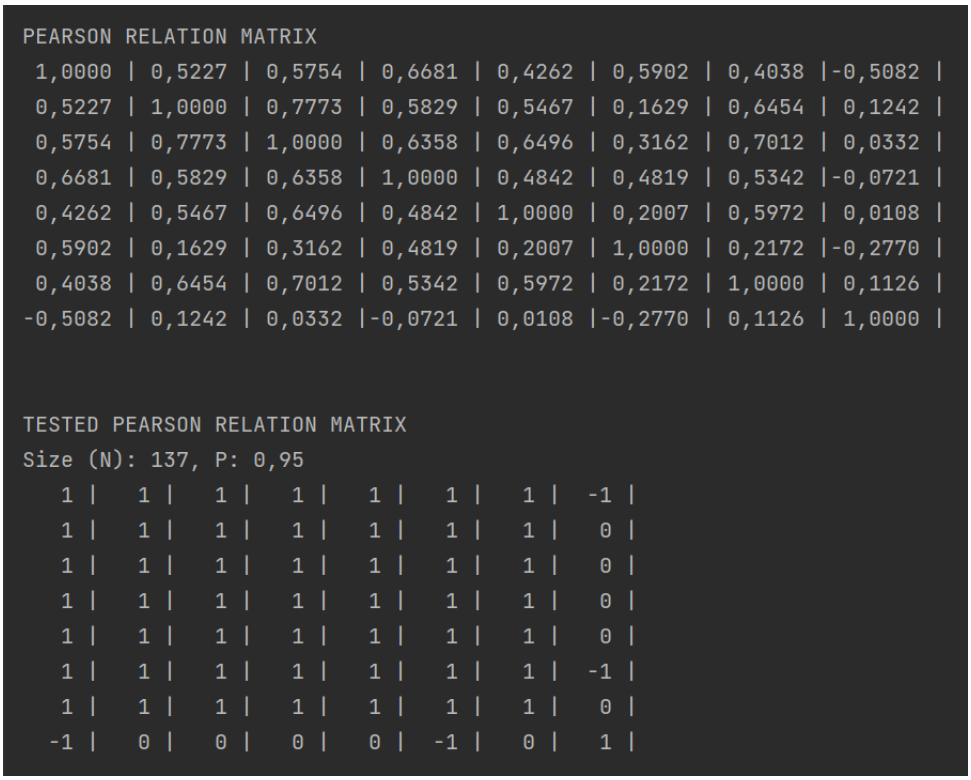
Вивід результатів на консоль:



EXPERIMENTAL DATA TABLE

Leukocytes	Lymphocytes	T-Lymphocytes	T-Helpers	RE-T-Suppressors	Sens. Theophylline	Res. Theophylline	B-Lymphocytes
6.0	1.2	0.7	0.53	0.39	0.34	0.42	0.31
6.0	1.46	0.99	0.58	0.6	0.3	0.71	0.39
7.9	2.45	1.29	1.35	0.71	0.78	0.86	0.39
12.2	3.02	2.23	1.42	0.91	0.61	2.1	0.23
5.3	1.17	0.63	0.46	0.28	0.36	0.7	0.23
6.1	1.2	0.91	0.59	0.48	0.2	0.48	0.33
5.8	1.06	0.57	0.51	0.28	0.4	0.65	0.13
4.6	2.85	1.27	1.06	0.46	0.02	1.07	0.33
6.7	2.45	1.62	1.07	0.55	0.78	1.32	0.55
4.4	3.01	1.75	0.65	0.65	0.03	1.11	0.54
14.6	3.47	2.93	1.68	1.51	0.51	1.79	0.15
5.7	1.42	0.73	0.52	0.48	0.4	0.52	0.33
7.5	1.95	1.17	1.78	0.69	0.47	0.94	0.39
4.4	3.11	1.5	0.99	0.42	0.05	1.47	0.33
4.9	2.12	1.46	1.06	0.91	0.05	1.37	0.49
5.0	1.4	0.91	0.52	0.32	0.28	0.7	0.21
8.0	2.24	1.51	0.87	0.67	0.4	0.94	0.31

Рисунок 1 – виводиться вся ТЕД, що була зчитана з файлу



PEARSON RELATION MATRIX

1,0000	0,5227	0,5754	0,6681	0,4262	0,5902	0,4038	-0,5082
0,5227	1,0000	0,7773	0,5829	0,5467	0,1629	0,6454	0,1242
0,5754	0,7773	1,0000	0,6358	0,6496	0,3162	0,7012	0,0332
0,6681	0,5829	0,6358	1,0000	0,4842	0,4819	0,5342	-0,0721
0,4262	0,5467	0,6496	0,4842	1,0000	0,2007	0,5972	0,0108
0,5902	0,1629	0,3162	0,4819	0,2007	1,0000	0,2172	-0,2770
0,4038	0,6454	0,7012	0,5342	0,5972	0,2172	1,0000	0,1126
-0,5082	0,1242	0,0332	-0,0721	0,0108	-0,2770	0,1126	1,0000

TESTED PEARSON RELATION MATRIX

Size (N): 137, P: 0,95

1	1	1	1	1	1	1	-1
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	-1
1	1	1	1	1	1	1	0
-1	0	0	0	0	-1	0	1

Рисунок 2 – матриця зв’язків Пірсона та матриця, що перевіряє важливість кожного коефіцієнта

Як бачимо на головній діагоналі завжди значення 1, тому що критерій порівнюється із самим собою

KENDALL RELATION MATRIX

1,0000	0,2168	0,2881	0,4000	0,2033	0,5104	0,1837	-0,3228
0,2168	1,0000	0,6080	0,4259	0,3954	0,0668	0,4610	0,1034
0,2881	0,6080	1,0000	0,4939	0,4673	0,1777	0,5294	0,0493
0,4000	0,4259	0,4939	1,0000	0,3818	0,2711	0,4013	0,0106
0,2033	0,3954	0,4673	0,3818	1,0000	0,1220	0,4483	0,0794
0,5104	0,0668	0,1777	0,2711	0,1220	1,0000	0,1167	-0,1934
0,1837	0,4610	0,5294	0,4013	0,4483	0,1167	1,0000	0,1149
-0,3228	0,1034	0,0493	0,0106	0,0794	-0,1934	0,1149	1,0000

TESTED KENDALL RELATION MATRIX

Size (N): 137, P: 0,95

1	1	1	1	1	1	1	-1
1	1	1	1	1	0	1	0
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	0
1	0	1	1	1	1	1	-1
1	1	1	1	1	1	1	1
-1	0	0	0	0	-1	1	1

Рисунок 3 – матриця зв’язків Кендала та матриця, що перевіряє важливість кожного коефіцієнта

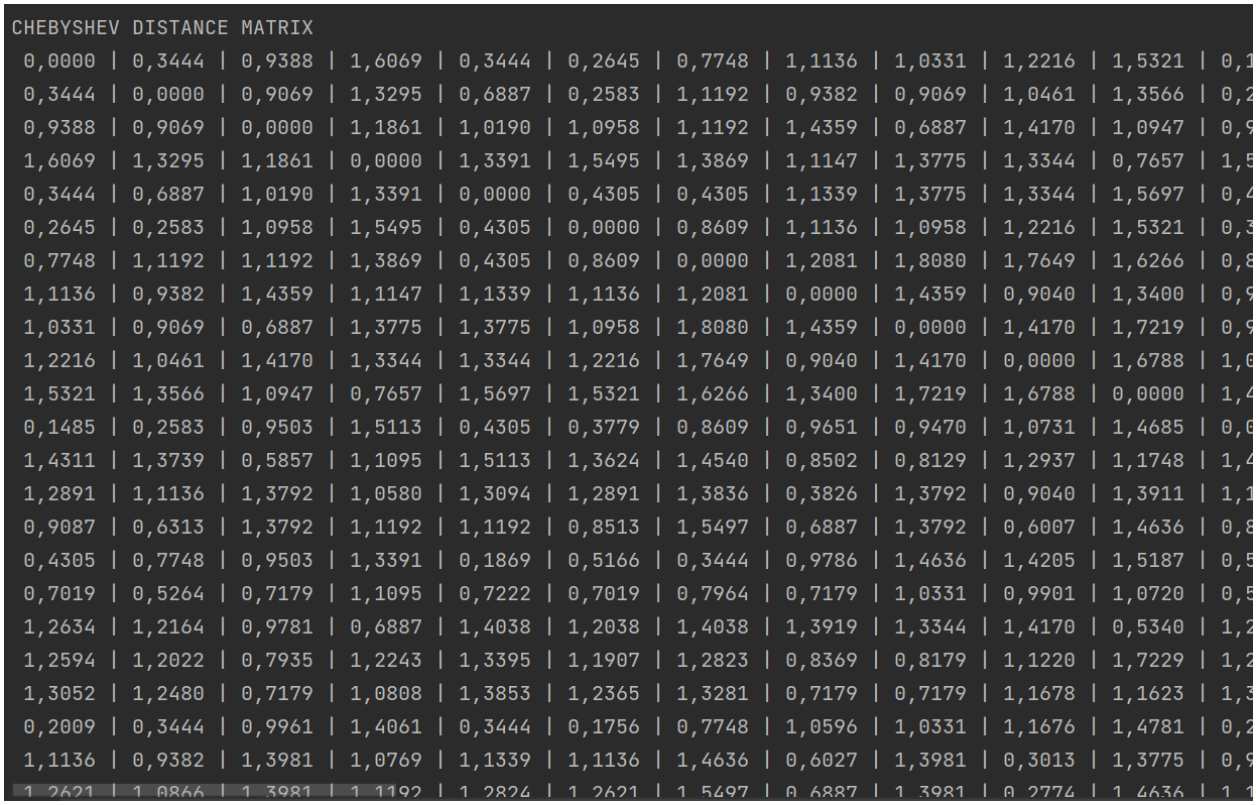
Як бачимо, останні матриці майже повністю співпадають для Пірсона та Кендала, окрім деяких порогових значень.

EXPERIMENTAL DATA TABLE NORMALIZED WITH UNIT CUBE

Leukocytes	Lymphocytes	T-Lymphocytes	T-Helpers	RE-T-Suppressors	Sens. Theophylline	Res. Theophylline	B-Lymphocytes
-0.21638	-0.75481	-0.48849	-0.55707	-0.45488	-0.05800	-0.60692	-0.03311
-0.21638	-0.57933	-0.29492	-0.49983	-0.18687	-0.13357	-0.32953	0.311254
0.021876	0.088844	-0.09467	0.381748	-0.04649	0.773285	-0.18606	0.311254
0.561098	0.473554	0.532760	0.461892	0.208749	0.452105	1.0	-0.37749
-0.30416	-0.77506	-0.53521	-0.63722	-0.59526	-0.02021	-0.33910	-0.37749
-0.20384	-0.75481	-0.34832	-0.48838	-0.34002	-0.32250	-0.54953	0.052975
-0.24146	-0.84930	-0.57526	-0.57997	-0.59526	0.055354	-0.38693	-0.80795
-0.39194	0.358816	-0.10802	0.049724	-0.36554	-0.66257	0.014801	0.052975
-0.12860	0.088844	0.125594	0.061173	-0.25068	0.773285	0.253927	1.0
-0.41702	0.466805	0.212367	-0.41968	-0.12306	-0.64368	0.053061	0.956953
0.862059	0.777273	1.0	0.759568	0.974475	0.263176	0.703483	-0.72186
-0.25400	-0.60633	-0.46846	-0.56852	-0.34002	0.055354	-0.51127	0.052975
-0.02828	-0.24862	-0.17477	0.874059	-0.07201	0.187605	-0.10954	0.311254
-0.41702	0.534298	0.045496	-0.03041	-0.41659	-0.60589	0.397402	0.052975
-0.35432	-0.13388	0.018796	0.049724	0.208749	-0.60589	0.301752	0.741720
-0.34178	-0.61983	-0.34832	-0.56852	-0.54421	-0.17136	-0.33910	-0.46358
0.034416	-0.05289	0.052171	-0.16780	-0.09754	0.055354	-0.10954	-0.03311
1.0	0.264326	0.466011	0.702323	0.808568	0.244283	0.311317	-0.33444

Рисунок 4 – центрована та нормалізована ТЕД за «одиничним кубом»

Там, де значення дорівнює 1, було максимальне значення стовпця для початкової ТЕД.



```
CHEBYSHEV DISTANCE MATRIX
 0,0000 | 0,3444 | 0,9388 | 1,6069 | 0,3444 | 0,2645 | 0,7748 | 1,1136 | 1,0331 | 1,2216 | 1,5321 | 0,1
 0,3444 | 0,0000 | 0,9069 | 1,3295 | 0,6887 | 0,2583 | 1,1192 | 0,9382 | 0,9069 | 1,0461 | 1,3566 | 0,2
 0,9388 | 0,9069 | 0,0000 | 1,1861 | 1,0190 | 1,0958 | 1,1192 | 1,4359 | 0,6887 | 1,4170 | 1,0947 | 0,9
 1,6069 | 1,3295 | 1,1861 | 0,0000 | 1,3391 | 1,5495 | 1,3869 | 1,1147 | 1,3775 | 1,3344 | 0,7657 | 1,5
 0,3444 | 0,6887 | 1,0190 | 1,3391 | 0,0000 | 0,4305 | 0,4305 | 1,1339 | 1,3775 | 1,3344 | 1,5697 | 0,4
 0,2645 | 0,2583 | 1,0958 | 1,5495 | 0,4305 | 0,0000 | 0,8609 | 1,1136 | 1,0958 | 1,2216 | 1,5321 | 0,3
 0,7748 | 1,1192 | 1,1192 | 1,3869 | 0,4305 | 0,8609 | 0,0000 | 1,2081 | 1,8080 | 1,7649 | 1,6266 | 0,8
 1,1136 | 0,9382 | 1,4359 | 1,1147 | 1,1339 | 1,1136 | 1,2081 | 0,0000 | 1,4359 | 0,9040 | 1,3400 | 0,9
 1,0331 | 0,9069 | 0,6887 | 1,3775 | 1,3775 | 1,0958 | 1,8080 | 1,4359 | 0,0000 | 1,4170 | 1,7219 | 0,9
 1,2216 | 1,0461 | 1,4170 | 1,3344 | 1,3344 | 1,2216 | 1,7649 | 0,9040 | 1,4170 | 0,0000 | 1,6788 | 1,0
 1,5321 | 1,3566 | 1,0947 | 0,7657 | 1,5697 | 1,5321 | 1,6266 | 1,3400 | 1,7219 | 1,6788 | 0,0000 | 1,4
 0,1485 | 0,2583 | 0,9503 | 1,5113 | 0,4305 | 0,3779 | 0,8609 | 0,9651 | 0,9470 | 1,0731 | 1,4685 | 0,0
 1,4311 | 1,3739 | 0,5857 | 1,1095 | 1,5113 | 1,3624 | 1,4540 | 0,8502 | 0,8129 | 1,2937 | 1,1748 | 1,4
 1,2891 | 1,1136 | 1,3792 | 1,0580 | 1,3094 | 1,2891 | 1,3836 | 0,3826 | 1,3792 | 0,9040 | 1,3911 | 1,1
 0,9087 | 0,6313 | 1,3792 | 1,1192 | 1,1192 | 0,8513 | 1,5497 | 0,6887 | 1,3792 | 0,6007 | 1,4636 | 0,8
 0,4305 | 0,7748 | 0,9503 | 1,3391 | 0,1869 | 0,5166 | 0,3444 | 0,9786 | 1,4636 | 1,4205 | 1,5187 | 0,5
 0,7019 | 0,5264 | 0,7179 | 1,1095 | 0,7222 | 0,7019 | 0,7964 | 0,7179 | 1,0331 | 0,9901 | 1,0720 | 0,5
 1,2634 | 1,2164 | 0,9781 | 0,6887 | 1,4038 | 1,2038 | 1,4038 | 1,3919 | 1,3344 | 1,4170 | 0,5340 | 1,2
 1,2594 | 1,2022 | 0,7935 | 1,2243 | 1,3395 | 1,1907 | 1,2823 | 0,8369 | 0,8179 | 1,1220 | 1,7229 | 1,2
 1,3052 | 1,2480 | 0,7179 | 1,0808 | 1,3853 | 1,2365 | 1,3281 | 0,7179 | 0,7179 | 1,1678 | 1,1623 | 1,3
 0,2009 | 0,3444 | 0,9961 | 1,4061 | 0,3444 | 0,1756 | 0,7748 | 1,0596 | 1,0331 | 1,1676 | 1,4781 | 0,2
 1,1136 | 0,9382 | 1,3981 | 1,0769 | 1,1339 | 1,1136 | 1,4636 | 0,6027 | 1,3981 | 0,3013 | 1,3775 | 0,9
 1,2621 | 1,0866 | 1,3981 | 1,1192 | 1,2824 | 1,2621 | 1,5497 | 0,6887 | 1,3981 | 0,2774 | 1,4636 | 1,1
```

Рисунок 5 – матриця відстані за методом Чебишева, на головній діагоналі присутні нулі.

Повна матриця має розмір NxN, тобто 137x137 і є також симетричною відносно головної діагоналі.

Висновок: вивчив принцип побудови таблиці експериментальних даних (ТЕД) типу «об’єкт-ознака» та методи їх попередньої обробки.