ДОДАТОК А

Вихідний код програми

```
     public class Model {
         private Map<String, Integer> diseasesIds; //map
disease - id
         private Map<String, Integer> symptomIds; //map symptom
- id
         private Map<Integer, Double> symptomsIdIdf; //map
symptom id - idf
         private int[][] disSymptBinaryLinks; //dis - row,
symptom - column, 1 - has link, 0 - doesn't
         private double[][] disSymptTfIdfLinks; //dis - row,
symptom - column, >0 - has link, 0 - doesn't

         private Model() {

         }

         public void setDiseasesIds(Map<String, Integer>
diseasesIds) {
             this.diseasesIds = diseasesIds;
         }

         public void setSymptomIds(Map<String, Integer>
symptomIds) {
             this.symptomIds = symptomIds;
         }

         public void setDisSymptBinaryLinks(int[][]
disSymptBinaryLinks) {
             this.disSymptBinaryLinks = disSymptBinaryLinks;
         }

         public Map<String, Integer> getDiseasesIds() {
             return diseasesIds;
         }

         public Map<String, Integer> getSymptomIds() {
             return symptomIds;
         }

         public void setDisSymptTfIdfLinks(double[][]
disSymptTfIdfLinks) {
             this.disSymptTfIdfLinks = disSymptTfIdfLinks;
         }

         //Logic

         //Testing
```

```
        // jaccard, correct rows, all features
        public double testSimpleJaccard() {
            Classifier classifier = new Classifier(new
JaccardSimilarity());

            int[] features = new int[symptomIds.size()];
            for (int i = 0; i < features.length; i++) {
                features[i] = i;
            }

            //id of test - id of result
            double correct = 0;
            for (int i = 0; i < disSymptBinaryLinks.length;
i++) {
                int[] row = disSymptBinaryLinks[i];
                int resultIndex = Util.getIndexOfLargest(
                        classifier.calculateSimilarities(
                                row,
                                disSymptBinaryLinks,
                                features
                        )
                );
                if (i == resultIndex) correct++;
            }
            return correct / diseasesIds.size();
        }

        // jaccard, rows with errors, customized features
        public double testRandomizedJaccard(int errorsAmnt,
int featuresAmnt) {
            Classifier classifier = new Classifier(new
JaccardSimilarity());
            int[][] customizedFeatures =
getCustomizedFeatures(disSymptBinaryLinks, featuresAmnt);

            //id of test - id of result
            double correct = 0;
            for (int i = 0; i < disSymptBinaryLinks.length;
i++) {
                int[] noisedDisease =
Arrays.copyOf(disSymptBinaryLinks[i],
disSymptBinaryLinks[i].length);
                if (errorsAmnt > 0) {
                    noiseDisease(noisedDisease,
customizedFeatures[i], errorsAmnt);
                }

                List<Integer> resultIndexes = Util.
                        getTopLargestIndexes(
                        classifier.calculateSimilarities(
                                noisedDisease,
```

```
                                    disSymptBinaryLinks,
                                    customizedFeatures[i]
                            ), 3);
                    int finalI = i;
                    if (resultIndexes.stream().anyMatch(v ->
v.equals(finalI))) correct++;
                }
                return correct / diseasesIds.size();
            }

            public double testSimpleCosineTfIdf() {
                Classifier classifier = new Classifier(new
CosineSimilarity());

                int[] features = new int[symptomIds.size()];
                for (int i = 0; i < features.length; i++) {
                    features[i] = i;
                }

                //id of test - id of result
                double correct = 0;
                for (int i = 0; i < disSymptTfIdfLinks.length;
i++) {
                    double[] row = disSymptTfIdfLinks[i]; //todo
calculate tf-idf for disease (query)
                    int resultIndex = Util.getIndexOfLargest(
                            classifier.calculateSimilarities(
                                    row,
                                    disSymptTfIdfLinks,
                                    features
                            )
                    );
                    if (i == resultIndex) correct++;
                }
                return correct / diseasesIds.size();
                return 0;
            }

            private int[] noiseDisease(int[] disease, int[]
featureVector, int noiseAmnt) {
                int[] history = new int[noiseAmnt];
                int i = 0;

                do {
                    int featureIndex =
featureVector[Util.getRandomNumberInRange(0,
featureVector.length - 1)];
                    if (i > 0 && Arrays.stream(history).anyMatch(v
-> featureIndex == v)) {
                        continue;
                    }
```

```
                disease[featureIndex] = disease[featureIndex]
== 0 ? 1 : 0;
                history[i++] = featureIndex;
            } while (i < noiseAmnt);

            return disease;
        }

        private int[][] getCustomizedFeatures(int[][]
disSymptLinks, int featuresAmnt) {
            int[][] customizedFeatures = new
int[disSymptLinks.length][featuresAmnt];

            for (int i = 0; i < disSymptLinks.length; i++) {
                int[] features = customizedFeatures[i];
                int[] row = disSymptLinks[i];
                int k = 0;

                long ones = Arrays.stream(row).filter(v -> v
== 1).count();
                Set<Integer> indexes = new HashSet<>();
                IntStream.range(0,
row.length).forEach(indexes::add);
                Iterator<Integer> iterator =
indexes.iterator();

                while (k < (int) (featuresAmnt * 0.8) && ones
> k) {
                    int j = iterator.next();

                    if (Arrays.stream(features).noneMatch(f ->
f == j) &&
                            row[j] == 0) {
                        continue;
                    }

                    features[k] = j;
                    k++;
                }

                iterator = indexes.iterator();
                while (k < featuresAmnt) {
                    int j = iterator.next();

                    if (Arrays.stream(features).noneMatch(f ->
f == j) &&
                            row[j] == 1) {
                        continue;
                    }

                    features[k] = j;
                    k++;
```

```
                }

                customizedFeatures[i] = features;
            }

            return customizedFeatures;
        }

        public static class Builder {
            private static String diseasesRes =
"diseases.txt";
            private static String symptomsRes =
"symptoms.txt";
            private static String linksRes = "disease-symptom-
TF_IDF-links [filtered].txt";

            public static Model build() throws IOException {
                Model model = new Model();

                DataParser dataParser = new DataParser();
                FileReader fileReader = new FileReader();



model.setDiseasesIds(dataParser.linesToIdMap(fileReader.readLin
es(diseasesRes)));

model.setSymptomIds(dataParser.linesToIdMap(fileReader.readLine
s(symptomsRes)));

model.setDisSymptBinaryLinks(dataParser.linksToBinaryMatrix(
                        model.getDiseasesIds(),
                        model.getSymptomIds(),
                        fileReader.readLines(linksRes)
                ));

model.setDisSymptTfIdfLinks(dataParser.linksToTfIdfMatrix(
                        model.getDiseasesIds(),
                        model.getSymptomIds(),
                        fileReader.readLines(linksRes)
                ));

                return model;
            }
        }

     public class Symptom {
    private int id;
    private String name;
    private double tf;
    private double idf;
    private double tf_idf;
```

```
    public Symptom(String name, double tf, double idf, double
tf_idf) {
        this.name = name;
        this.tf = tf;
        this.idf = idf;
        this.tf_idf = tf_idf;
    }

    @Override
    public String toString() {
        return name + "\t" +
                tf + "\t" +
                idf + "\t" +
                tf_idf;
    }

    public static Symptom parse(String[] linkArray) {
        //linkArray[0] - disease
        return new Symptom(
                linkArray[1],
                Double.parseDouble(linkArray[2]),
                Double.parseDouble(linkArray[3]),
                Double.parseDouble(linkArray[4])
        );
    }

}
```

```
 import java.util.Arrays;
import java.util.List;
import java.util.Map;

import static java.util.stream.Collectors.toMap;

public class DataParser {
    public Map<String, Integer> linesToIdMap(List<String>
lines) {
        //map entity - id
        return lines.stream().skip(1).map(line ->
line.split("\t"))
                .collect(toMap(arr -> arr[1], arr ->
Integer.parseInt(arr[0])));
    }

    public int[][] linksToBinaryMatrix(Map<String, Integer>
diseases,
                                       Map<String, Integer>
symptoms,
                                       List<String> links) {
        int[][] disSymptLinks = new
```

```
int[diseases.entrySet().size()][symptoms.entrySet().size()];
        links.stream().skip(1)
                .map(line -> line.split("\t"))
                .forEach(arr ->
disSymptLinks[diseases.get(arr[0])][symptoms.get(arr[1])] = 1);
        return disSymptLinks;
    }

    public double[][] linksToTfIdfMatrix(Map<String, Integer>
diseases,
                                         Map<String, Integer>
symptoms,
                                         List<String> links) {
        double[][] disSymptLinks = new
double[diseases.entrySet().size()][symptoms.entrySet().size()];
        links.stream().skip(1)
                .map(line -> line.split("\t"))
                .forEach(arr ->
disSymptLinks[diseases.get(arr[0])][symptoms.get(arr[1])] =
Double.parseDouble(arr[4]));
        return disSymptLinks;
    }
}


     import similarity.Similarity;

import java.util.stream.IntStream;

public class Classifier {
    private Similarity similarity;

    public Classifier(Similarity similarity) {
        this.similarity = similarity;
    }

    public double[] calculateSimilarities(int[] input, int[][]
data, int[] features) {
        double[] similarities = new double[data.length];
        IntStream.range(0, data.length).parallel()
                .forEach(i -> similarities[i] =
similarity.calculate(input, data[i], features));
        return similarities;
    }
}

     package similarity;

public class CosineSimilarity implements Similarity {
    @Override
    public double calculate(int[] inputA, int[] inputB, int[]
features) {
```

```
        return 0;
    }
}
```

```
package similarity;

import java.util.Arrays;
import java.util.stream.IntStream;

public class JaccardSimilarity implements Similarity {
    /***
     * Jaccard similarity = p/p+q+r
     * p - number of features that positive for both objects
     * q - number of features that positive for object A and
negative for object B
     * r - number of features that positive for object B and
negative for object A
     *
     * @param inputA - features vector of object A
     * @param inputB - features vector of object B
     * @param features - indexes of features we need to
calculate
     * @return similarity between inputA and inputB based on
features
     */
    @Override
    public double calculate(int[] inputA, int[] inputB, int[]
features) {
        double p = 0, q = 0, r = 0;
        for (int fIndex : features) {
            p += (inputA[fIndex] + inputB[fIndex] == 2) ? 1 :
0;
            q += (inputA[fIndex] > inputB[fIndex]) ? 1 : 0;
            r += (inputA[fIndex] < inputB[fIndex]) ? 1 : 0;
        }
        return p / (p + q + r);
    }
}
```

```
package similarity;

public interface Similarity {
    double calculate(int[] inputA, int[] inputB, int[]
features);
}
```

ДОДАТОК Б

Відомість атестаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	69 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Дудар В.В.			Дослідження ансамбля алгоритмів класифікації у підсистемі симптоматичної діагностики захворювань.	Шифр групи	Код напр./спец.
Перевірив		Магдаліна І.В				СШІм-19-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					