Raspodijeljena obrada velike količine podataka

3. Domaća zadaća

Filip Gulan - JMBAG: 0036479428

1. zadatak

1. U izlaznoj datoteci se nalazi 8357 različitih zapisa.
2. Najsličnija šala šali s ID-jem 1 je šala s ID-jem 87 (0.44775397).
3. U obe šale riječ `doctor` se pojavila više puta te također dijele još nekoliko istih riječi.
4. Da, imat će smisla, jer možemo vidjeti na prethodnom primjeru da je preporučio dvije slične šale.

Izvorni kod

Task1.java

```
public class Task1 {

    private static final String INPUT_FILE =
"/Users/filipgulan/Downloads/jester_dataset_2/jester_items.dat";
    private static final String OUTPUT_FILE = "item_similarity.csv";

    private static StandardAnalyzer analyzer = new StandardAnalyzer();
    private static Directory index = new RAMDirectory();

    public static void main(String[] args) throws IOException,
ParseException {
        List<String> lines =  Files.readAllLines(Paths.get(INPUT_FILE));
        Map<Integer, String> jokePairs = parseInput(lines);
        createLuceneDocuments(jokePairs);
        float[][] similarityMatrix = queryDocs(jokePairs);
        similarityMatrix = normalizeSimilarityMatrix(similarityMatrix);
        storeSimilarityMatrixAsCSV(similarityMatrix, OUTPUT_FILE);
    }

    private static void storeSimilarityMatrixAsCSV(float[][]
similarityMatrix, String path) throws IOException {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(path))) {
            for (int i = 0; i < similarityMatrix.length; ++i) {
                for (int j = i + 1; j < similarityMatrix[i].length; ++j) {
                    if (similarityMatrix[i][j] > 0) {
```

```
                        String row = (i + 1) + "," + (j + 1) + "," +
similarityMatrix[i][j];
                        writer.write(row);
                        if (i == similarityMatrix.length - 1 && j ==
similarityMatrix[i].length -1) continue;
                        writer.newLine();
                    }
                }
            }
        }
    }

    private static float[][] normalizeSimilarityMatrix(float[][]
similarityMatrix) {
        for (Integer i = 0; i < similarityMatrix.length; i++) {
            Float max = similarityMatrix[i][i];
            for (Integer j = 0; j < similarityMatrix[i].length; j++) {
                similarityMatrix[i][j] /= max;
            }
        }

        for (int i = 0; i < similarityMatrix.length; i ++) {
            for (int j = 0; j <= i; j++) {
                similarityMatrix[j][i] = similarityMatrix[i][j] =
                        (similarityMatrix[i][j] + similarityMatrix[j][i]) /
2.0f;

            }
        }
        return similarityMatrix;
    }

    private static float[][] queryDocs(Map<Integer, String> jokePairs)
throws ParseException, IOException {
        Integer documentsCount = jokePairs.size();
        float[][] similarity = new float[documentsCount][documentsCount];
        IndexReader reader = DirectoryReader.open(index);
        IndexSearcher searcher = new IndexSearcher(reader);

        Integer currentDocumentIndex = 0;
        for (Map.Entry<Integer, String> entry : jokePairs.entrySet()) {
            Query query = new QueryParser("text",
analyzer).parse(QueryParser.escape(entry.getValue()));
            for (ScoreDoc hit : searcher.search(query,
documentsCount).scoreDocs) {
                Document document = reader.document(hit.doc);
                Integer documentID = Integer.parseInt(document.get("ID"));
                Integer documentIndex = documentID - 1;
                similarity[currentDocumentIndex][documentIndex] = hit.score;
```

```
            }
            currentDocumentIndex++;
        }
        reader.close();
        return similarity;
    }

    private static void createLuceneDocuments(Map<Integer, String>
jokePairs) throws IOException {
        FieldType idFieldType = new FieldType();
        idFieldType.setStored(true);
        idFieldType.setTokenized(false);
        idFieldType.setIndexOptions(IndexOptions.NONE);

        FieldType jokeFieldType = new FieldType();
        jokeFieldType.setStored(true);
        jokeFieldType.setTokenized(true);

jokeFieldType.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS);

        IndexWriterConfig config = new IndexWriterConfig(analyzer);
        IndexWriter w = new IndexWriter(index, config);
        for (Map.Entry<Integer, String> entry : jokePairs.entrySet()) {
            Document document = new Document();
            document.add(new Field("ID", entry.getKey().toString(),
idFieldType));
            document.add(new Field("text", entry.getValue(),
jokeFieldType));
            w.addDocument(document);
        }
        w.close();
    }

    private static Map<Integer, String> parseInput(List<String> lines) {
        Iterator<String> iterator = lines.iterator();
        Map<Integer, String> jokes = new HashMap<>();

        while (iterator.hasNext()) {
            Integer id = Integer.parseInt(iterator.next().replace(":",
"").trim());
            String joke = "";
            while (iterator.hasNext()) {
                String line = iterator.next();
                if (line.isEmpty()) {
                    break;
                }
                joke += line;
            }
```

```
            joke =
StringEscapeUtils.unescapeXml(joke.toLowerCase().replaceAll("\\<.*?\\>",
""));
            jokes.put(id, joke);
        }
        return jokes;
    }
}
```

2. zadatak

1. Preporučitelj temeljen na sličnosti objekata dao je sljedeću silaznu listu (prvi element je onaj kojeg sustav je preporučio prvog, zatim drugog i tako dalje): (36, 43, 96, 22, 37, 42, 94, 122, 86, 129), dok je preporučitelj temeljen na suradnji korisnika dao sljedeću listu: (105, 89, 53, 35, 32, 72, 104, 129, 114, 108).
2. Preporučitelj temeljen na sličnosti objekata je nešto bolji od preporučitelj temeljenog na suradnji korisnika (4.614624 vs. 4.938738 - Pearson) - manja brojka znači kvalitetniji procjenitelj.
3. Bolje je koristiti mjeru log-likelihood jer daje nešto bolje rezultate.

Izvorni kod

ItemBasedEvaluator.java

```
public class ItemBasedEvaluator {

    private static final String SIMILARITY_FILE =

"/Users/filipgulan/Diplomski/Semester_8/ROVKP/ROVKP_DZ3/item_similarity.csv"
;
    private static final String MODEL_FILE =

"/Users/filipgulan/Downloads/jester_dataset_2/jester_ratings.dat";

  public static void main(String[] args) throws IOException, TasteException
{
        RandomUtils.useTestSeed();
        DataModel model = new FileDataModel(new File(MODEL_FILE), "\\s+");
        RecommenderBuilder builder = model1 -> {
            ItemSimilarity similarity = new FileItemSimilarity(new
File(SIMILARITY_FILE));
            return new GenericItemBasedRecommender(model1, similarity);
        };
        RecommenderEvaluator recEvaluator = new RMSRecommenderEvaluator();
        double score = recEvaluator.evaluate(builder, null, model, 0.4,
0.6);
        System.out.println(score);
    }
}
```

ItemBasedJokeRecommender.java

```
public class ItemBasedJokeRecommender {

    private static final String SIMILARITY_FILE =

"/Users/filipgulan/Diplomski/Semester_8/ROVKP/ROVKP_DZ3/item_similarity.csv"
;
    private static final String MODEL_FILE =

"/Users/filipgulan/Downloads/jester_dataset_2/jester_ratings.dat";

    public static void main(String[] args) throws IOException,
TasteException {
        DataModel model = new FileDataModel(new File(MODEL_FILE), "\\s+");
        ItemSimilarity similarity = new FileItemSimilarity(new
File(SIMILARITY_FILE));
        ItemBasedRecommender recommender = new
GenericItemBasedRecommender(model, similarity);

        List<RecommendedItem> recommendations = recommender.recommend(220,
10);
        for (RecommendedItem recommendation : recommendations) {
            System.out.println(recommendation);
        }
    }
}
```

UserBasedEvaluator.java

```
    private static final String MODEL_FILE =

"/Users/filipgulan/Downloads/jester_dataset_2/jester_ratings.csv";

    public static void main(String[] args) throws IOException,
TasteException {
        RandomUtils.useTestSeed();
        DataModel model = new FileDataModel(new File(MODEL_FILE));
        RecommenderBuilder builder = model1 -> {
//            UserSimilarity similarity = new
PearsonCorrelationSimilarity(model1);
            UserSimilarity similarity = new LogLikelihoodSimilarity(model);
            UserNeighborhood neighborhood = new
ThresholdUserNeighborhood(0.1, similarity, model1);
            return new GenericUserBasedRecommender(model1, neighborhood,
similarity);
        };

        RecommenderEvaluator recEvaluator = new RMSRecommenderEvaluator();
        double score = recEvaluator.evaluate(builder, null, model, 0.4,
0.6);
        System.out.println(score);
    }
}
```

UserBasedJokeRecommender.java

```
public class UserBasedJokeRecommender {

    private static final String MODEL_FILE =

"/Users/filipgulan/Downloads/jester_dataset_2/jester_ratings.csv";

    public static void main(String[] args) throws IOException,
TasteException {
        DataModel model = new FileDataModel(new File(MODEL_FILE));
        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1,
similarity, model);
        UserBasedRecommender recommender = new
GenericUserBasedRecommender(model, neighborhood, similarity);;

        List<RecommendedItem> recommendations = recommender.recommend(220,
10);
        for (RecommendedItem recommendation : recommendations) {
            System.out.println(recommendation);
        }
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter("out.txt"))) {
            for (int id = 1; id <101; id++) {
                try {
                    String row = Integer.toString(id) + "\t[";

                    recommendations = recommender.recommend(id, 10);
                    for (RecommendedItem recommendation : recommendations) {
                        row += Long.toString(recommendation.getItemID()) +
":" + Float.toString(recommendation.getValue()) + ",";
                    }
                    row += "]";
                    writer.write(row);
                    writer.newLine();
                } catch (TasteException excp) {
                    continue;
                }
            }
        }
    }
}
```

3. zadatak

1. Izvršeno je 9 MapReduce poslova.
2. Da, postoje razlike u izlaznim datotekama i izračunatim preporukama, i to vjerojatno zbog internog algoritma koji se izvršava kod `recommenditembased` naredbe.

Niz naredbi:

```
$ hdfs dfs -put jester_ratings.csv /dz3/
$ hdfs dfs -put users.txt /dz3/
$ mahout recommenditembased --similarityClassname
SIMILARITY_PEARSON_CORRELATION --input /dz3/jester_ratings.csv --usersFile
/dz3/users.txt --output /dz3/result --numRecommendations 10
```