|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №5**

**«Mahout. Система рекомендаций»**

**ДИСЦИПЛИНА: «Технологии обработки больших данных»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-72Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Голубева С.Е. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2023

**Цель:** формирование практических навыков работы с библиотекой Mahout для создания рекомендательных систем на основе больших данных.

**Задачи:**

1. Изучить алгоритмы системы рекомендаций на основе коллаборативной фильтрации.
2. Научиться реализовывать системы рекомендаций с помощью Apache Mahout
3. Научиться выполнять оценку правильности работы системы рекомендаций.

**Задание:**

Для выполнения задания использовать базу данных MovieLens любого размера:

<https://grouplens.org/datasets/movielens/>

Реализовать 2 системы рекомендаций фильмов (по варианту) для пользователя на основе его оценок. В системах, в которых используются метрики, реализовать как минимум 2 версии с применением разных метрик. Сравнить оценки правильности работы всех систем. Для сравнения запускать алгоритм оценки как минимум 10 раз и использовать среднее значение оценки для каждой из систем.

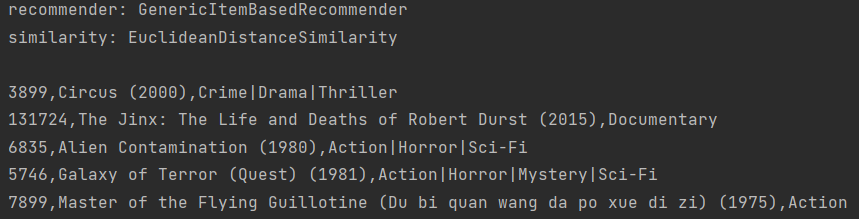
**Вариант 7**

* TreeClusteringRecommender. Реализовать как минимум 2 версии с различными метриками
* SlopeOneRecommender

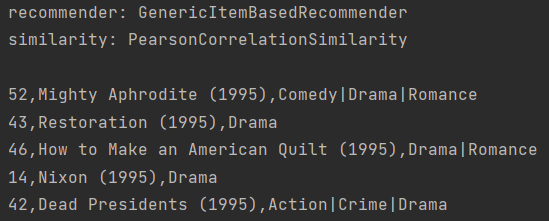
**Листинг:**

package org.example;  
  
import org.apache.mahout.cf.taste.common.TasteException;  
import org.apache.mahout.cf.taste.eval.RecommenderBuilder;  
import org.apache.mahout.cf.taste.eval.RecommenderEvaluator;  
import org.apache.mahout.cf.taste.impl.eval.AverageAbsoluteDifferenceRecommenderEvaluator;  
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;  
import org.apache.mahout.cf.taste.impl.recommender.\*;  
import org.apache.mahout.cf.taste.impl.recommender.slopeone.SlopeOneRecommender;  
import org.apache.mahout.cf.taste.impl.similarity.\*;  
import org.apache.mahout.cf.taste.model.DataModel;  
import org.apache.mahout.cf.taste.recommender.RecommendedItem;  
import org.apache.mahout.cf.taste.recommender.Recommender;  
import org.apache.mahout.cf.taste.similarity.ItemSimilarity;  
  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.\*;  
  
public class Main {  
 public static void main(String args[]) throws IOException, TasteException {  
 File file = new File("D:\\movies.csv");  
 FileReader fr = new FileReader(file);  
 BufferedReader br = new BufferedReader(fr);  
  
 Map<Long, String> movies = new HashMap<Long, String>();  
 String line;  
 while((line = br.readLine()) != null){  
 movies.put(Long.*parseLong*(line.split(",")[0]), line);  
 }  
  
 DataModel model = new FileDataModel(new File("D:\\ratings.csv"));ItemSimilarity similarity\_1 = new EuclideanDistanceSimilarity(model);  
 GenericItemBasedRecommender recommender\_1 = new GenericItemBasedRecommender(model, similarity\_1);  
 List<RecommendedItem> recommendations\_1 = recommender\_1.recommend(1, 5);  
 System.*out*.println("recommender: GenericItemBasedRecommender");  
 System.*out*.println("similarity: EuclideanDistanceSimilarity");  
 System.*out*.println();  
 for (RecommendedItem recommendation : recommendations\_1) {  
 System.*out*.println(movies.get(recommendation.getItemID()));  
 }  
 System.*out*.println();  
  
 ItemSimilarity similarity\_2 = new PearsonCorrelationSimilarity(model);  
 GenericItemBasedRecommender recommender\_2 = new GenericItemBasedRecommender(model, similarity\_2);  
 List<RecommendedItem> recommendations\_2 = recommender\_2.recommend(1, 5);  
 System.*out*.println("recommender: GenericItemBasedRecommender");  
 System.*out*.println("similarity: PearsonCorrelationSimilarity");  
 System.*out*.println();  
 for (RecommendedItem recommendation : recommendations\_2) {  
 System.*out*.println(movies.get(recommendation.getItemID()));  
 }  
 System.*out*.println();  
  
 SlopeOneRecommender recommender\_3 = new SlopeOneRecommender(model);  
 List<RecommendedItem> recommendations\_3 = recommender\_3.recommend(1, 5);  
 System.*out*.println("recommender: SlopeOneRecommender");  
 System.*out*.println();  
 for (RecommendedItem recommendation : recommendations\_3) {  
 System.*out*.println(movies.get(recommendation.getItemID()));  
 }  
 System.*out*.println();  
  
 RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();  
 RecommenderBuilder builder\_1 = new RecommenderBuilder() {  
 @Override  
 public Recommender buildRecommender(DataModel model) throws TasteException {  
 ItemSimilarity similarity = new EuclideanDistanceSimilarity(model);  
 return new GenericItemBasedRecommender (model, similarity);  
 }  
 };  
 double score\_1 = 0;  
 for(int i = 0; i < 10; i++) {  
 double value = evaluator.evaluate(builder\_1, null, model, 0.7, 0.2);  
 score\_1 += value;  
 }  
 score\_1 /= 10;  
 System.*out*.println("GenericItemBasedRecommender, EuclideanDistanceSimilarity: " + score\_1);  
  
 RecommenderBuilder builder\_2 = new RecommenderBuilder() {  
 @Override  
 public Recommender buildRecommender(DataModel model) throws TasteException {  
 ItemSimilarity similarity = new PearsonCorrelationSimilarity(model);  
 return new GenericItemBasedRecommender (model, similarity);  
 }  
 };  
 double score\_2 = 0;  
 for(int i = 0; i < 10; i++) {  
 double value = evaluator.evaluate(builder\_2, null, model, 0.7, 0.2);  
 score\_2 += value;  
 }  
 score\_2 /= 10;  
 System.*out*.println("GenericItemBasedRecommender, PearsonCorrelationSimilarity: " + score\_2);  
  
 RecommenderBuilder builder\_3 = new RecommenderBuilder() {  
 @Override  
 public Recommender buildRecommender(DataModel model) throws TasteException {  
 return new SlopeOneRecommender(model);  
 }  
 };  
 double score\_3 = 0;  
 for(int i = 0; i < 10; i++) {  
 double value = evaluator.evaluate(builder\_3, null, model, 0.7, 0.2);  
 score\_3 += value;  
 }  
 score\_3 /= 10;  
 System.*out*.println("SlopeOneRecommender: " + score\_3);  
 }  
}

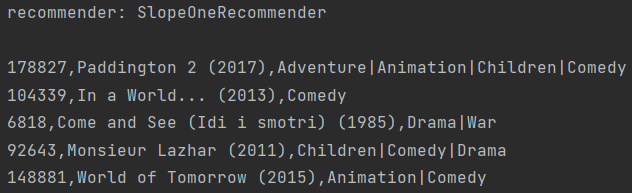
**Результат:**



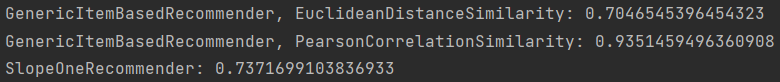
**Рис. 1.** Результат для GenericItemBasedRecommender, EuclideanDistanceSimilarity



**Рис. 2.** Результат для GenericItemBasedRecommender, PearsonCorrelationSimilarity



**Рис. 3.** Результат для SlopeOneRecommender



**Рис. 4.** Оценки правильности работы систем

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки работы с библиотекой Mahout для создания рекомендательных систем на основе больших данных.