

Informe Práctica de Mahout

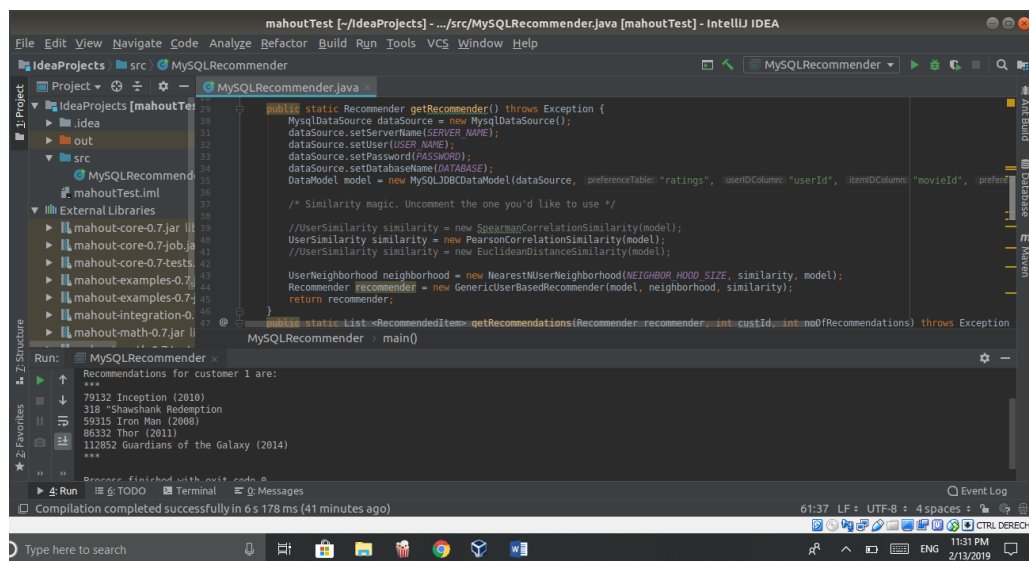
En clase hemos venido hablando sobre el mundo del Big Data y de las herramientas comunes que se utilizan para su manejo. Las plataformas digitales que manejan productos muchas veces están interesadas en hacer que su aplicación aprenda de los gustos de su comunidad de usuarios para poder ofrecerles recomendaciones similares. Para esta práctica se pide utilizar los algoritmos de similitud que ofrece Mahout, con la finalidad de tomar un dataset de películas y las valoraciones que le dieron distintos usuarios a éstas, de tal manera que podamos ofrecer recomendaciones de otras películas a dicho usuario. Mahout es importante para nosotros debido a que está montado sobre Hadoop, y utiliza su mecanismo de MapReduce, herramientas de las cuales se habló ampliamente en clase.

Para introducirnos en el funcionamiento de Mahout, primero se realizó una práctica guiada con libros, en la cual se tomaban varios pasos para probar el algoritmo. Primero, desde MySQL, se crearon las tablas que iban a contener la información de los libros, la identificación del cliente y la valoración que dicho cliente le otorgó a tal libro. Una vez eso configurado, se realizó una aplicación Java en la cual se utilizarían las librerías de Mahout para poder ejecutar los algoritmos de similitud tomando los datos de la información almacenada en MySQL.

Con los conocimientos básicos de la estructura de código para utilizar Mahout, se procedió a realizar la práctica de verdad. Utilizando un dataset descargado de internet, el cual contiene información de una gran cantidad de películas, se importó su contenido en tablas de MySQL para ser utilizadas en la aplicación Java posteriormente. En cuanto al uso

de las funciones para recomendación, se tomó como código base el realizado en el tutorial, haciendo las debidas adaptaciones para el nuevo dataset. Las funciones que se utilizaron fueron las de distancia euclidiana, algoritmo de Pearson y algoritmo de Spearman. Estos fueron los resultados:

- Pearson



```
public static Recommender getRecommender() throws Exception {
    MySQLDataSource dataSource = new MySQLDataSource();
    dataSource.setServerName(SERVER_NAME);
    dataSource.setUser(USER_NAME);
    dataSource.setPassword(PASSWORD);
    dataSource.setDatabaseName(DATABASE);
    DataModel model = new MySQLJDBCDataModel(dataSource, "ratings", "userId", "movieId", "rating");

    /* Similarity magic. Uncomment the one you'd like to use */
    //UserSimilarity similarity = new SpearmanCorrelationSimilarity(model);
    UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    //UserSimilarity similarity = new EuclideanDistanceSimilarity(model);

    UserNeighborhood neighborhood = new NearestUserNeighborhood(NEIGHBOR_HOOD_SIZE, similarity, model);
    Recommender recommender = new GenericUserBasedRecommender(model, neighborhood, similarity);
    return recommender;
}

public static List<RecommendedItem> getRecommendations(Recommender recommender, int custId, int noOfRecommendations) throws Exception {
    MySQLRecommender recommender = new MySQLRecommender();
    return recommender.getRecommendations(custId, noOfRecommendations);
}
```

Run: MySQLRecommender x

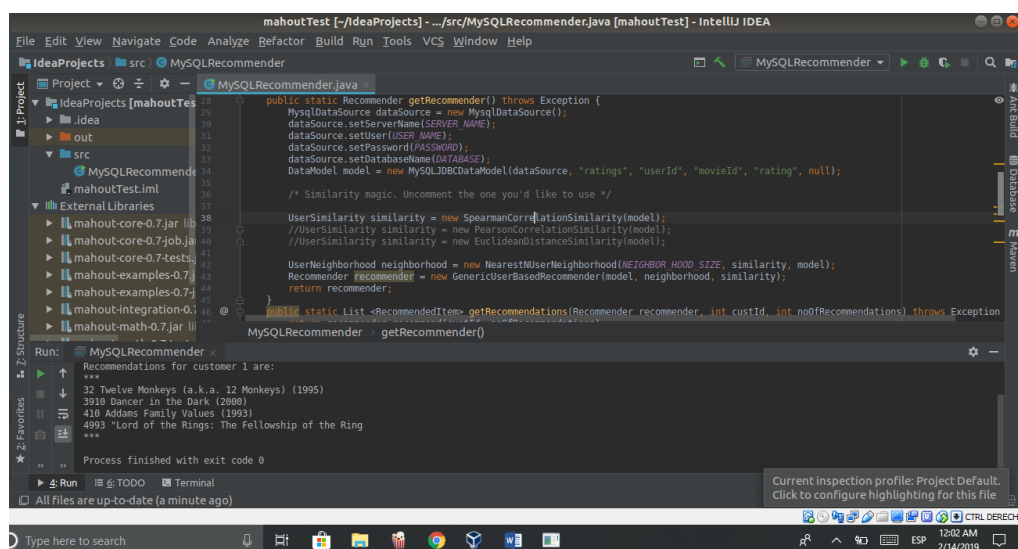
Recommendations for customer 1 are:

- 79132 Inception (2010)
- 318 Shawshank Redemption
- 59315 Iron Man (2008)
- 85332 Thor (2011)
- 112852 Guardians of the Galaxy (2014)

Process finished with exit code 0

Compilation completed successfully in 6 s 178 ms (41 minutes ago)

- Spearman



```
public static Recommender getRecommender() throws Exception {
    MySQLDataSource dataSource = new MySQLDataSource();
    dataSource.setServerName(SERVER_NAME);
    dataSource.setUser(USER_NAME);
    dataSource.setPassword(PASSWORD);
    dataSource.setDatabaseName(DATABASE);
    DataModel model = new MySQLJDBCDataModel(dataSource, "ratings", "userId", "movieId", "rating", null);

    /* Similarity magic. Uncomment the one you'd like to use */
    UserSimilarity similarity = new SpearmanCorrelationSimilarity(model);
    //UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
    //UserSimilarity similarity = new EuclideanDistanceSimilarity(model);

    UserNeighborhood neighborhood = new NearestUserNeighborhood(NEIGHBOR_HOOD_SIZE, similarity, model);
    Recommender recommender = new GenericUserBasedRecommender(model, neighborhood, similarity);
    return recommender;
}

public static List<RecommendedItem> getRecommendations(Recommender recommender, int custId, int noOfRecommendations) throws Exception {
    MySQLRecommender recommender = new MySQLRecommender();
    return recommender.getRecommendations(custId, noOfRecommendations);
}
```

Run: MySQLRecommender x

Recommendations for customer 1 are:

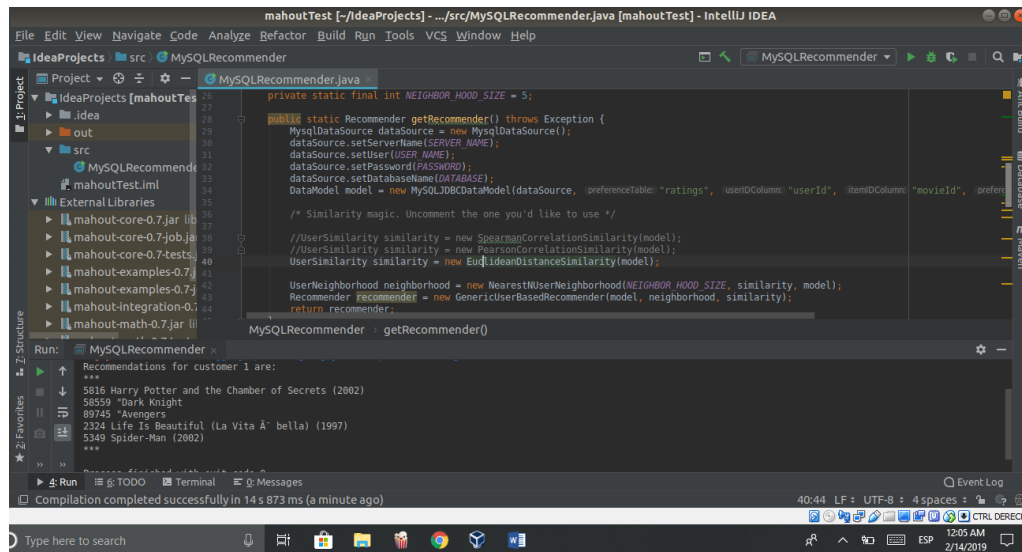
- 32 Twelve Monkeys (a.k.a. 12 Monkeys) (1995)
- 3910 Dancer in the Dark (2000)
- 410 Addams Family Values (1993)
- 4993 Lord of the Rings: The Fellowship of the Ring

Process finished with exit code 0

All Files are up-to-date (a minute ago)

Current inspection profile: Project Default. Click to configure highlighting for this file

- Euclidean



La diferencia entre cada algoritmo es la siguiente: En el caso de Pearson y Spearman, ocurre un cálculo entre dos variables para indicar su cercanía o similitud expresada en un rango de 1 a -1, solo que los dos trabajan enfocados a dos tipos de variables diferentes. Dependen de si son métricas u ordinales. La distancia euclidiana lo que hace es obtener la cercanía de gustos entre dos variables, las más lejos no se tomarán en cuenta para la recomendación.