Movie Recommendation System using Apache Spark deployed on EMR Cluster

A **Recommendation System** (that is, **Recommendation Engine or RE**) is an information filtering systems that helps predict the rating based on the ratings given by other users to an item. There are a couple of ways to develop recommendation engines that typically produce a list of recommendations.

- **Content-based filtering approach:**

Using content-based filtering approach, a series of discrete characteristics of an item is utilized to recommend additional items with similar properties. This approach tries to recommend items that are similar to those that a user liked in the past. A key issue with content-based filtering is whether the system is able to learn user preferences from users' actions regarding one content source and use them across other content types (Cold Start). The other problems are scalability and sparsity.

- **Model-based collaborative filtering approach:**

In the model-based collaborative filtering technique, users and products are described by a small set of factors, also called latent factors (LFs). The LFs are then used to predict the missing entries. The Alternating Least Squares (ALS) algorithm is used to learn these LFs. This filtering approach is commonly used for real-time movie recommendations.

Dataset: The dataset is downloaded from MovieLens. It contains 25,000,095 ratings from 162,541 users on 62,423 movies.

A. Apache Spark based Recommendation System:

Spark RDD and Spark SQL are used for exploratory data analysis, data cleaning and transformations required to create model. Spark MLlib provides Matrix Factorization Model and Alternating Least Squares algorithm. The code is written in the native Spark language Scala. The Scala application JAR file is deployed on Elastic MapReduce cluster which provides a managed Hadoop framework that is easy, fast and cost-effective to process vast amounts of data across EC2 instances.

- **IntelliJ Project Setup Information:**

- *JDK Version:* 1.8.0_271-b09 - *Scala Version*: 2.11.12 - *SBT Version*: 1.14.0
Check the SBT Build file on right to see Spark project dependencies.
sbt-assembly is used to create a fat JAR of project with all its dependencies.

- **AWS EMR Cluster Setup Information:**

- *Software Configuration: Release:* emr-5.31.0
Applications: Spark 2.4.6 on Hadoop 2.10.0 YARN and Zeppelin 0.8.2
- *Hardware Configuration: Instance Type:* 4 General Purpose m5.xlarge
# of instances: 4 (1 Master Node and 3 Core Nodes)

```
name := "spark_als_aws"

version := "0.1"

scalaVersion := "2.11.12"

// https://mvnrepository.com/artifact/org.apache.spark/spark-core
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.4.6" % "provided"

// https://mvnrepository.com/artifact/org.apache.spark/spark-sql
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.4.6" % "provided"

// https://mvnrepository.com/artifact/org.apache.spark/spark-mllib
libraryDependencies += "org.apache.spark" %% "spark-mllib" % "2.4.6" % "compile"

mainClass in (assembly) := Some("alsrecommendation")
assemblyJarName in assembly := "spark_ml_aws.jar"
```

```
assemblyMergeStrategy in assembly := {
  case PathList("META-INF", xs @ _*) =>
    xs map {_.toLowerCase} match {
      case "manifest.mf" :: Nil | "index.list" :: Nil | "dependencies" :: Nil =>
        MergeStrategy.discard
      case ps @ x :: xs if ps.last.endsWith(".sf") || ps.last.endsWith(".dsa") =>
        MergeStrategy.discard
      case "plexus" :: xs =>
        MergeStrategy.discard
      case "services" :: xs =>
        MergeStrategy.filterDistinctLines
      case "spring.schemas" :: Nil | "spring.handlers" :: Nil =>
        MergeStrategy.filterDistinctLines
      case _ => MergeStrategy.first
    }
  case "application.conf" => MergeStrategy.concat
  case "reference.conf" => MergeStrategy.concat
  case _ => MergeStrategy.first
```

B. Item-based collaborative filtering for movie similarity:

Two similarity metrics are used in form of Cosine similarity based on Euclidean Distance metric and Jaccard similarity based on set theory. These two metrics find the similarities between two movies based on the ratings received by similar users. The model outputs the top 10 movies recommendations for a given movie.

```
 * The cosine similarity between two vectors A, B is
 *   dotProduct(A, B) / (norm(A) * norm(B))
 */
def cosineSimilarity(dotProduct: Double, ratingNorm: Double, rating2Norm: Double): Double = {
  dotProduct / (ratingNorm * rating2Norm)
}
```

```
 * The Jaccard Similarity between two sets A, B is
 *  |Intersection(A, B)| / |Union(A, B)|
 */
def jaccardSimilarity(usersInCommon: Double, totalUsers1: Double, totalUsers2: Double): Double = {
  val union = totalUsers1 + totalUsers2 - usersInCommon
  usersInCommon / union
```

Results: Die Hard (1988)

Movie Name	Movie Suggestion	Correlation	Regularized Correlation	Cosine Similarity	Jaccard Similarity
Die Hard (1988)	Fearless (1993)	-0.7674	-0.4338	0.8622	0.0549
Die Hard (1988)	How to Make an American Quilt (1995)	-0.5106	-0.3919	0.9856	0.1284
Die Hard (1988)	Rich Man's Wife, The (1996)	-0.8887	-0.3626	0.9109	0.0303
Die Hard (1988)	Flubber (1997)	-0.5881	-0.3619	0.8638	0.0632
Die Hard (1988)	His Girl Friday (1940)	-0.5411	-0.3608	0.9478	0.0778
Die Hard (1988)	Believers, The (1987)	-0.8416	-0.3466	0.8789	0.0302
Die Hard (1988)	Crow: City of Angels, The (1996)	-0.4753	-0.3268	0.8896	0.0913
Die Hard (1988)	Barb Wire (1996)	-0.4983	-0.3265	0.8632	0.0805
Die Hard (1988)	Ninotchka (1939)	-0.6143	-0.3218	0.8956	0.0464
Die Hard (1988)	Goofy Movie, A (1995)	-0.8528	-0.3198	0.8540	0.0253

Seven (Se7en) (1995)

Movie Name	Movie Suggestion	Correlation	Regularized Correlation	Cosine Similarity	Jaccard Similarity
Seven (Se7en) (1995)	Specialist, The (1994)	-0.6532	-0.3692	0.8369	0.0583
Seven (Se7en) (1995)	Thirty-Two Short Films About Glenn Gould (1993)	-0.7321	-0.3468	0.8921	0.0400
Seven (Se7en) (1995)	Heaven & Earth (1993)	-0.7267	-0.3442	0.8751	0.0415
Seven (Se7en) (1995)	Phantom, The (1996)	-0.4430	-0.3423	0.8679	0.1318
Seven (Se7en) (1995)	Blue in the Face (1995)	-0.5416	-0.3410	0.8763	0.0700
Seven (Se7en) (1995)	Scout, The (1994)	-0.7454	-0.3313	0.9043	0.0364
Seven (Se7en) (1995)	Sweet Hereafter, The (1997)	-0.5466	-0.3280	0.8677	0.0638
Seven (Se7en) (1995)	Drop Dead Fred (1991)	-0.6777	-0.3210	0.7689	0.0391
Seven (Se7en) (1995)	Out to Sea (1997)	-0.6838	-0.3039	0.8063	0.0354
Seven (Se7en) (1995)	Little Odessa (1994)	-0.7308	-0.3009	0.7807	0.0318

C. Model-based recommendation to find movies based on user preferences:

Data exploration using Spark SQL:

```
// Get the max, min ratings along with the count of users who have rated a movie.
val results = spark.sql(
  "SELECT movies.title, movierates.maxr, movierates.minr, "
    + "movierates.cntu FROM (SELECT ratings.movieId,max(ratings.rating) AS maxr, "
    + "MIN(ratings.rating) AS minr, COUNT(distinct userId) AS cntu "
    + "FROM ratings GROUP BY ratings.movieId) movierates "
    + "JOIN movies ON movierates.movieId = movies.movieId "
    + "ORDER BY movierates.cntu DESC ")
```

```
+--------------------------------------------------+----+----+-----+
|title                                             |maxr|minr|cntu |
+--------------------------------------------------+----+----+-----+
|Forrest Gump (1994)                               |5.0 |0.5 |81491|
|Shawshank Redemption, The (1994)                  |5.0 |0.5 |81482|
|Pulp Fiction (1994)                               |5.0 |0.5 |79672|
|Silence of the Lambs, The (1991)                  |5.0 |0.5 |74127|
|Matrix, The (1999)                                |5.0 |0.5 |72674|
```

ALS Model Parameters:

numBlocks: Number of blocks to parallelize (-1 is auto-configure). Chosen = -1
rank: Number of LFs. Chosen = 10
iterations: Max iterations of ALS to run. Chosen = 10
lambda: Regularization parameter. Chosen = 0.1
implicitPrefs: True is explicit feedback and false is implicit feedback. Chosen = False
alpha: Parameter Governing Baseline confidence in preference observations. (only implicit feedback). Chosen = 1.0

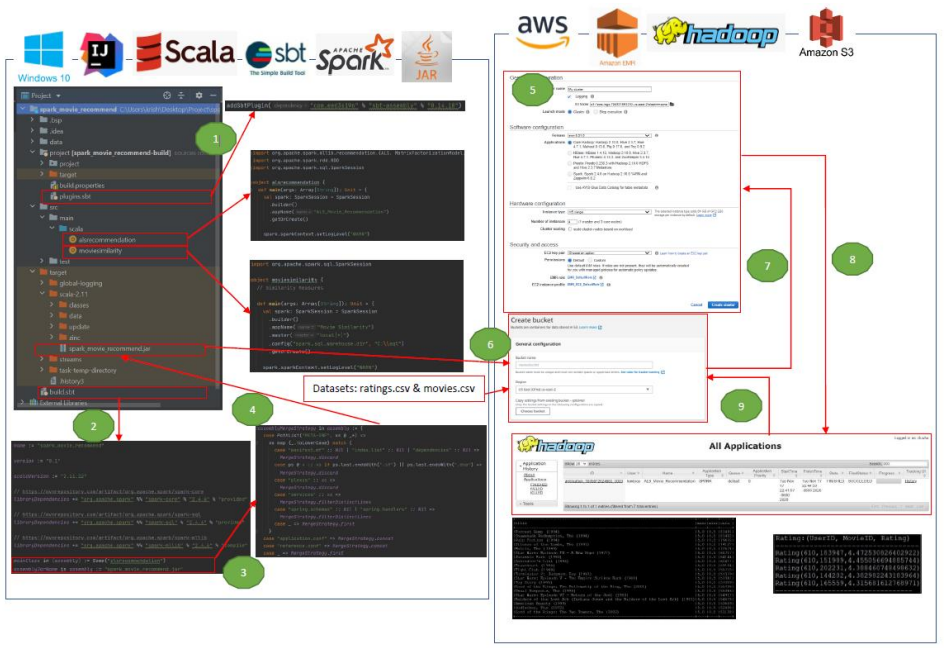
Results:

```
Test RMSE = 0.8089947016113231
```

Top 5 recommendations for user 610:

```
Rating:(UserID, MovieID, Rating)
--------------------------------
Rating(610,183947,4.472530826402922)
Rating(610,151989,4.455056694885744)
Rating(610,202231,4.388460748498632)
Rating(610,144202,4.382982243183964)
Rating(610,165559,4.315681612768971)
--------------------------------
```

D. Deployment Workflow:



E. References:

- Dataset - Learning Material - Code Base - sbt-assembly - Assembly Strategy - AWS EMR Tutorial