**package** module4

**import** breeze.linalg.DenseVector

**import** scala.language.implicitConversions

**import** scala.io.Source

**import** java.io.PrintWriter

**import** java.io.FileWriter

**object** UserUserRecommender {

//Code added just to support unset values on the linear algebra framework (Breeze)

**class** RichDenseVector(vector : DenseVector[Double]){

**def** asSetArray = vector.toArray.filterNot(x => x.isNaN())

**def** asSetVector = DenseVector(vector.toArray.filterNot(x => x.isNaN()))

**def** transformUnSetValue = DenseVector(vector.toArray.map(x => **if**(x.isNaN()) 0 **else** x))

**def** mean = {

**val** v = asSetArray

v.sum / v.size

}

}

//Code added just to support unset values on the linear algebra framework (Breeze)

**implicit** **def** richDenseVector(vector : DenseVector[Double]) = **new** RichDenseVector(vector)

**def** recommender(moviesPath : String, usersPath : String, ratingsPath : String, usersMovies : List[(Int, Int)]) : List[(Int, Int, Double, String)] ={

//read the movies file

**val** movies = Source.fromFile(getClass().getResource("movie-titles.csv").getPath()).getLines.toList.map(x => x.split(",")).map(x => (x(0).toInt, x(1))).zipWithIndex.map(x => (x.\_1.\_1, (x.\_1.\_2, x.\_2))).toMap

//read the users file and create a map with id user as key and empty Vector as value

**val** vectors = Source.fromFile(getClass().getResource("users.csv").getPath()).getLines.toList.map(x => x.split(",")).map(x => (x(0).toInt, DenseVector.fill(movies.size){Double.NaN})).toMap

//read the ratings file and fill the vector of each user (with the mapping from the movies)

Source.fromFile(getClass().getResource("ratings.csv").getPath()).getLines.toList.map(x => x.split(",")).foreach({x =>

vectors(x(0).toInt)(movies(x(1).toInt).\_2) = x(2).toDouble

})

// for each query, launch the recommender

**for**(userMovie <- usersMovies) **yield** (userMovie.\_1, userMovie.\_2, recommender(userMovie.\_1, userMovie.\_2, movies, vectors), movies(userMovie.\_2).\_1)

}

**def** recommender(user : Int, movie : Int, movies : Map[Int, (String, Int)], vectors : Map[Int, DenseVector[Double]]) : Double = {

//mean of the user

**val** userMean = vectors(user).mean

//mean centered all the vector to perform the recommender

**val** meanCentered = vectors.map(x => (x.\_1, x.\_2 - x.\_2.mean)).toMap

// compute the cosine similarity and take the 30st neighbors whose have already rated the item

**val** sim = meanCentered.filterNot(x => x.\_1 == user || x.\_2(movies(movie).\_2).isNaN() ).map(v => (v.\_1, (meanCentered(user).transformUnSetValue.dot(v.\_2.transformUnSetValue))/(meanCentered(user).transformUnSetValue.norm(2) \* v.\_2.transformUnSetValue.norm(2)))).toList.sortBy(\_.\_2 \* -1).take(30).toMap

//compute numerator S(u,v).(Rv,i - Uv)

**val** v = meanCentered.filterKeys(sim.keySet.contains(\_)).map(x => (x.\_1, sim(x.\_1) \* x.\_2(movies(movie).\_2)))

//compute denominator |S(u,v)| and sum the numerator and rounding

//(the isNaN check is not necessary here because only neighbor whose have already rated the item are taken, but I tried with other test cases than these from Courses)

BigDecimal(userMean + (v.map(\_.\_2).filterNot(\_.isNaN).sum /v.map(x => **if** (x.\_2.isNaN()) 0 **else** sim(x.\_1)).sum)).setScale(4, BigDecimal.RoundingMode.HALF\_UP).toDouble

}

**def** deliverable(fileName : String, usersMovies : List[(Int, Int)])

{

**val** recommendations = recommender(getClass().getResource("movie-titles.csv").getPath(),

getClass().getResource("movie-titles.csv").getPath(),

getClass().getResource("movie-titles.csv").getPath(),

usersMovies)

**val** writer = **new** PrintWriter(**new** FileWriter("/projects/Coursera-IntroductionToRecommenderSystems/Module4/ProgrammingAssignment3/" + fileName))

recommendations.foreach({x =>

writer.println(x.\_1 + "," + x.\_2 + "," + x.\_3 + "," + x.\_4)

})

writer.close()

}

}