package MovieRec ;

import java.io.\* ;

import java.util.\* ;

import org.apache.commons.cli.\* ;

import org.apache.commons.configuration.\* ;

import org.apache.commons.csv.\* ;

public

class Main

{

static PropertiesConfiguration config ;

static boolean isToShow = false ;

static String configFilePath = "config.properties" ;

public static

void main (String [] args)

{

Options options = new Options() ;

options.addOption("c", "config", true, "configuration file") ;

options.addOption("d", "display", false, "show statistics") ;

options.addOption("h", "help", false, "show help message") ;

CommandLineParser parser = new DefaultParser() ;

CommandLine cmd = null ;

try {

cmd = parser.parse(options, args) ;

if (cmd.hasOption("d"))

isToShow = true ;

if (cmd.hasOption("c"))

configFilePath = cmd.getOptionValue("c") ;

if (cmd.hasOption("h")) {

HelpFormatter formater = new HelpFormatter() ;

formater.printHelp("Usage", options) ;

System.exit(0) ;

}

}

catch (ParseException e) {

System.err.println(e) ;

System.exit(1) ;

}

config(configFilePath) ;

try {

MovieRatingData data = new MovieRatingData(config) ;

FileReader ftrain = new FileReader(config.getString("data.training")) ;

FileReader ftest = new FileReader(config.getString("data.testing")) ;

data.load(ftrain) ;

if (isToShow)

data.show() ;

data.removeOutliers() ;

Recommender rec = new Recommender(config) ;

rec.train(data) ;

test(ftest, rec) ;

}

catch (IOException e) {

System.err.println(e) ;

System.exit(1) ;

}

}

public static void config(String fpath) {

try {

config = new PropertiesConfiguration(fpath) ;

}

catch (ConfigurationException e) {

System.err.println(e) ;

System.exit(1) ;

}

}

public static void test(FileReader ftest, Recommender rec) throws IOException

{

int [][] error = new int[2][2] ; // actual x predict -> #

TreeMap<Integer, HashSet<Integer>>

users = new TreeMap<Integer, HashSet<Integer>>();

TreeMap<Integer, HashSet<Integer>>

q\_positive = new TreeMap<Integer, HashSet<Integer>>();

TreeMap<Integer, HashSet<Integer>>

q\_negative = new TreeMap<Integer, HashSet<Integer>>();

for (CSVRecord r : CSVFormat.newFormat(',').parse(ftest)) {

Integer user = Integer.parseInt(r.get(0)) ;

Integer movie = Integer.parseInt(r.get(1)) ;

Double rating = Double.parseDouble(r.get(2)) ;

String type = r.get(3) ;

if (users.containsKey(user) == false) {

users.put(user, new HashSet<Integer>()) ;

q\_positive.put(user, new HashSet<Integer>()) ;

q\_negative.put(user, new HashSet<Integer>()) ;

}

if (type.equals("c")) {

if (rating >= config.getDouble("data.like\_threshold"))

users.get(user).add(movie) ;

}

else /\* r.get(3) is "q" \*/{

if (rating >= config.getDouble("data.like\_threshold"))

q\_positive.get(user).add(movie) ;

else

q\_negative.get(user).add(movie) ;

}

}

for (Integer u : users.keySet()) {

HashSet<Integer> u\_movies = users.get(u) ;

for (Integer q : q\_positive.get(u))

error[1][rec.predict(u\_movies, q)] += 1 ;

for (Integer q : q\_negative.get(u))

error[0][rec.predict(u\_movies, q)] += 1 ;

}

System.out.print("Precision: ") ;

if (error[0][1] + error[1][1] > 0)

System.out.println( String.format("%.3f",

(double)(error[1][1]) / (double)(error[0][1] + error[1][1]))) ;

else

System.out.println("undefined.") ;

System.out.print("Recall: ") ;

if (error[1][0] + error[1][1] > 0)

System.out.println( String.format("%.3f",

((double)(error[1][1]) / (double)(error[1][0] + error[1][1])))) ;

else

System.out.println("undefined.") ;

System.out.print("All case accuracy: ") ;

if (error[0][0] + error[1][1] > 0)

System.out.println( String.format("%.3f",

((double)(error[1][1] + error[0][0]) /

(double)(error[0][0] + error[0][1] + error[1][0] + error[1][1])))) ;

else

System.out.println("undefined.") ;

System.out.println("[[" + error[0][0] + ", " + error[0][1] + "],") ;

System.out.println(" [" + error[1][0] + ", " + error[1][1] + "]]") ;

}

}

package MovieRec ; import java.io.\* ; import java.util.\* ;

import com.google.common.collect.\* ;

import org.apache.commons.configuration.\* ;

public class Recommender

{

TreeMap<Integer, Integer>

support1 = new TreeMap<Integer, Integer>() ;

/\* support1 : MovieId -> Num \*/

TreeMap<IntPair, Integer>

support2 = new TreeMap<IntPair, Integer>() ;

/\* support2 : MovieId x MovieId -> Num \*/

TreeMap<IntTriple, Integer>

support3 = new TreeMap<IntTriple, Integer>() ;

/\* support3 : MovieId x MovieId x MovieId -> Num \*/

PropertiesConfiguration config ;

int min\_supports ;

int min\_evidence\_3 ;

double threshold\_2 ;

double threshold\_3 ;

Recommender(PropertiesConfiguration config) {

this.config = config ;

this.min\_supports = config.getInt("training.min\_supports") ;

this.threshold\_2 = config.getDouble("prediction.threshold\_2") ;

this.threshold\_3 = config.getDouble("prediction.threshold\_3") ;

this.min\_evidence\_3 = config.getInt("prediction.min\_evidence\_3") ;

}

public void train(MovieRatingData data) {

TreeMap<Integer, HashSet<Integer>>

Baskets = data.getBaskets() ;

/\* Baskets : UserID -> Set<MovieId> \*/

for (Integer user : Baskets.keySet()) {

HashSet<Integer> Basket = Baskets.get(user) ;

updateSupport1(Basket) ;

updateSupport2(Basket) ;

updateSupport3(Basket) ;

}

}

public int predict(HashSet<Integer> profile, Integer q) {

if (predictPair(profile, q) == 1)

return 1 ;

return predictTriple(profile, q) ;

}

private void updateSupport1(HashSet<Integer> Basket) {

for (Integer item : Basket) {

Integer c = support1.get(item) ;

if (c == null)

c = new Integer(1) ;

else

c = new Integer(c.intValue() + 1 ) ;

support1.put(item, c) ;

}

}

private void updateSupport2(HashSet<Integer> Basket) {

if (Basket.size() >= 2) {

for (Set<Integer> pair : Sets.combinations(Basket, 2)) {

Integer c = support2.get(new IntPair(pair)) ;

if (c == null)

c = new Integer(1) ;

else

c = new Integer(c.intValue() + 1) ;

support2.put(new IntPair(pair), c) ;

}

}

}

private void updateSupport3(HashSet<Integer> Basket) {

HashSet<Integer>

\_Basket = new HashSet<Integer>() ;

for (Integer elem : Basket) {

if (support1.get(elem) >= min\_supports)

\_Basket.add(elem) ;

}

Basket = \_Basket ;

if (Basket.size() >= 3) {

for (Set<Integer> triple : Sets.combinations(Basket, 3)) {

Integer c = support3.get(new IntTriple(triple));

if (c == null)

c = new Integer(1) ;

else

c = new Integer(c.intValue() + 1) ;

support3.put(new IntTriple(triple), c) ;

}

}

}

private int predictPair(HashSet<Integer> profile, Integer q) {

/\* TODO: implement this method \*/

return 0 ;

}

private int predictTriple(HashSet<Integer> profile, Integer q) {

if (profile.size() < 2)

return 0 ;

int evidence = 0 ;

for (Set<Integer> p : Sets.combinations(profile, 2)) {

Integer den = support2.get(new IntPair(p)) ;

if (den == null)

continue ;

TreeSet<Integer> t = new TreeSet<Integer>(p) ;

t.add(q) ;

IntTriple item = new IntTriple(t) ;

Integer num = support3.get(item) ;

if (num == null)

continue ;

if (num.intValue() < min\_supports)

continue ;

if ((double)num / (double)den >= threshold\_3)

evidence++ ;

}

if (evidence >= min\_evidence\_3)

return 1 ;

return 0 ;

}

}

class IntPair implements Comparable

{

int first ;

int second ;

public IntPair(int first, int second) {

if (first <= second) {

this.first = first ;

this.second = second ;

}

else {

this.first = second;

this.second = first ;

}

}

public IntPair(Set<Integer> s) {

Integer [] elem = s.toArray(new Integer[2]) ;

if (elem[0] < elem[1]) {

this.first = elem[0] ;

this.second = elem[1] ;

}

else {

this.first = elem[1] ;

this.second = elem[0] ;

}

}

public int compareTo(Object obj) {

IntPair p = (IntPair) obj ;

if (this.first < p.first)

return -1 ;

if (this.first > p.first)

return 1 ;

return (this.second - p.second) ;

}

}

class IntTriple implements Comparable

{

int [] elem ;

IntTriple(Set<Integer> s) {

/\* TODO: implement this method \*/

}

public int compareTo(Object obj) {

/\* TODO: implement this method \*/

return 0 ;

}

}

package MovieRec ;

import java.io.\* ;

import java.util.\* ;

import java.awt.\* ;

import javax.swing.\* ;

import org.apache.commons.csv.\* ;

import org.apache.commons.configuration.\* ;

import org.jfree.chart.\* ;

import org.jfree.chart.plot.\* ;

import org.jfree.chart.renderer.xy.XYDotRenderer ;

import org.jfree.data.\* ;

import org.jfree.data.statistics.\* ;

import org.jfree.data.xy.XYDataset ;

import org.jfree.ui.ApplicationFrame ;

public

class MovieRatingData

{

TreeMap<Integer, HashSet<Integer>>

Baskets = new TreeMap<Integer, HashSet<Integer>>() ;

TreeMap<Integer, Integer>

numRatingsOfMovies = new TreeMap<Integer, Integer>() ;

TreeMap<Integer, Double>

accRatingsOfMovies = new TreeMap<Integer, Double>() ;

PropertiesConfiguration config ;

double like\_threshold ;

int outlier\_threshold ;

public MovieRatingData (PropertiesConfiguration config) {

this.config = config ;

this.like\_threshold = config.getDouble("data.like\_threshold") ;

this.outlier\_threshold = config.getInt("data.outlier\_threshold") ;

}

public void load (FileReader f) throws IOException {

for (CSVRecord r : CSVFormat.newFormat(',').parse(f)) {

Integer user = Integer.parseInt(r.get(0)) ;

Integer movie = Integer.parseInt(r.get(1)) ;

Double rating = Double.parseDouble(r.get(2)) ;

if (numRatingsOfMovies.containsKey(movie) == false) {

numRatingsOfMovies.put(movie, 1) ;

accRatingsOfMovies.put(movie, rating) ;

}

else {

numRatingsOfMovies.put(movie, numRatingsOfMovies.get(movie) + 1) ;

accRatingsOfMovies.put(movie, accRatingsOfMovies.get(movie) + rating) ;

}

if (rating >= like\_threshold) {

HashSet<Integer> basket = Baskets.get(user) ;

if (basket == null) {

basket = new HashSet<Integer>() ;

Baskets.put(user, basket) ;

}

basket.add(movie) ;

}

}

}

public void removeOutliers() {

HashSet<Integer> outliers = new HashSet<Integer>() ;

for (Integer userId : Baskets.keySet()) {

HashSet<Integer> basket = Baskets.get(userId) ;

if (basket.size() > outlier\_threshold)

outliers.add(userId) ;

}

for (Integer userId : outliers)

Baskets.remove(userId) ;

}

public TreeMap<Integer, HashSet<Integer>>

getBaskets() {

return Baskets ;

}

public void show() {

showMovieStat() ;

showUserStat() ;

showRatingStat() ;

}

private void showMovieStat() {

ApplicationFrame frame = new ApplicationFrame("Movie Stat.") ;

XYDataset dataset = getNumAvgRatingDataset() ;

JFreeChart chart = ChartFactory.createScatterPlot("Num vs. Avg Rating", "Num", "Avg Rating", dataset, PlotOrientation.VERTICAL, true, true, false) ;

XYPlot plot = (XYPlot) chart.getPlot() ;

XYDotRenderer renderer = new XYDotRenderer() ;

renderer.setDotWidth(2) ;

renderer.setDotHeight(2) ;

plot.setRenderer(renderer) ;

JPanel panel = new ChartPanel(chart) ;

panel.setPreferredSize(new java.awt.Dimension(500, 270)) ;

frame.setContentPane(panel) ;

frame.pack() ;

frame.setVisible(true) ;

}

private XYDataset getNumAvgRatingDataset() {

return (XYDataset) new NumAvgDataset(numRatingsOfMovies, accRatingsOfMovies) ;

}

private void showUserStat() {

ApplicationFrame frame = new ApplicationFrame("User Stat.") ;

double [] ratings = new double[Baskets.keySet().size()] ;

int i = 0 ;

for (Integer user : Baskets.keySet()) {

ratings[i] = (double) Baskets.get(user).size() ;

i++ ;

}

HistogramDataset dataset = new HistogramDataset() ;

dataset.setType(HistogramType.RELATIVE\_FREQUENCY) ;

dataset.addSeries("Histogram", ratings, 20) ;

JFreeChart chart = ChartFactory.createHistogram("Num. Ratings by Users", "Num", "value", dataset, PlotOrientation.VERTICAL, false, false, false) ;

JPanel panel = new ChartPanel(chart) ;

frame.setContentPane(panel) ;

frame.pack() ;

frame.setVisible(true) ;

}

private void showRatingStat() {

/\* TODO:

implement this method to draw a histogram

that shows the distribution of ratings (1.0~5.0)

\*/

}

}

package MovieRec ;

import org.jfree.data.DomainInfo;

import org.jfree.data.Range;

import org.jfree.data.RangeInfo;

import org.jfree.data.xy.AbstractXYDataset;

import org.jfree.data.xy.XYDataset;

import java.util.\* ;

public class NumAvgDataset extends AbstractXYDataset

implements XYDataset, DomainInfo, RangeInfo {

private Double[] num\_ratings; /\*\* The x values. \*/

private Double[] avg\_ratings; /\*\* The y values. \*/

private int seriesCount = 1 ; /\*\* The number of series. \*/

private int itemCount; /\*\* The number of items. \*/

private Number domainMin; /\*\* The minimum domain value. \*/

private Number domainMax; /\*\* The maximum domain value. \*/

private Number rangeMin; /\*\* The minimum range value. \*/

private Number rangeMax; /\*\* The maximum range value. \*/

private Range domainRange; /\*\* The range of the domain. \*/

private Range range; /\*\* The range. \*/

/\*\*

\* Creates a sample dataset using default settings (4 series, 100 data items per series,

\* random data in the range 0 - 200).

\*/

NumAvgDataset() { }

/\*\*

\* Creates a sample dataset.

\* @param seriesCount the number of series.

\* @param itemCount the number of items.

\*/

NumAvgDataset(TreeMap<Integer, Integer> num, TreeMap<Integer, Double> acc) {

this.num\_ratings = new Double[num.keySet().size()] ;

this.avg\_ratings = new Double[num.keySet().size()] ;

this.itemCount = num.keySet().size();

this.seriesCount = 1;

int max = 0 ;

int i = 0 ;

for(Integer movie : num.keySet()) {

int n ;

double a ;

n = num.get(movie) ;

a = acc.get(movie) / n ;

num\_ratings[i] = (double) n ;

avg\_ratings[i] = a ;

if (n > max)

max = n ;

i += 1 ;

}

this.domainMin = new Double(0);

this.domainMax = new Double(max);

this.domainRange = new Range(0, max);

this.rangeMin = 0.0 ;

this.rangeMax = 5.0 ;

this.range = new Range(0.0, 5.0);

}

/\*\*

\* Returns the x-value for the specified series and item. Series are numbered 0, 1, ...

\* @param series the index (zero-based) of the series.

\* @param item the index (zero-based) of the required item.

\* @return the x-value for the specified series and item.

\*/

public Number getX(int series, int item) {

return this.num\_ratings[item] ;

}

/\*\*

\* Returns the y-value for the specified series and item. Series are numbered 0, 1, ...

\* @param series the index (zero-based) of the series.

\* @param item the index (zero-based) of the required item.

\* @return the y-value for the specified series and item.

\*/

public Number getY(int series, int item) {

return this.avg\_ratings[item] ;

}

/\*\*

\* Returns the number of series in the dataset.

\* @return the series count.

\*/

public int getSeriesCount() {

return this.seriesCount;

}

/\*\*

\* Returns the key for the series.

\* @param series the index (zero-based) of the series.

\* @return The key for the series.

\*/

public Comparable getSeriesKey(int series) {

return "Sample " + series;

}

/\*\*

\* Returns the number of items in the specified series.

\* @param series the index (zero-based) of the series.

\* @return the number of items in the specified series.

\*/

public int getItemCount(int series) {

return this.itemCount;

}

/\*\*

\* Returns the minimum domain value.

\* @return The minimum domain value.

\*/

public double getDomainLowerBound() {

return this.domainMin.doubleValue();

}

/\*\*

\* Returns the lower bound for the domain.

\* @param includeInterval include the x-interval?

\* @return The lower bound.

\*/

public double getDomainLowerBound(boolean includeInterval) {

return this.domainMin.doubleValue();

}

/\*\*

\* Returns the maximum domain value.

\* @return The maximum domain value.

\*/

public double getDomainUpperBound() {

return this.domainMax.doubleValue();

}

/\*\*

\* Returns the upper bound for the domain.

\* @param includeInterval include the x-interval?

\* @return The upper bound.

\*/

public double getDomainUpperBound(boolean includeInterval) {

return this.domainMax.doubleValue();

}

/\*\*

\* Returns the range of values in the domain.

\* @return the range.

\*/

public Range getDomainBounds() {

return this.domainRange;

}

/\*\*

\* Returns the bounds for the domain.

\* @param includeInterval include the x-interval?

\* @return The bounds.

\*/

public Range getDomainBounds(boolean includeInterval) {

return this.domainRange;

}

/\*\*

\* Returns the range of values in the domain.

\* @return the range.

\*/

public Range getDomainRange() {

return this.domainRange;

}

/\*\*

\* Returns the minimum range value.

\* @return The minimum range value.

\*/

public double getRangeLowerBound() {

return this.rangeMin.doubleValue();

}

/\*\*

\* Returns the lower bound for the range.

\* @param includeInterval include the y-interval?

\* @return The lower bound.

\*/

public double getRangeLowerBound(boolean includeInterval) {

return this.rangeMin.doubleValue();

}

/\*\*

\* Returns the maximum range value.

\* @return The maximum range value.

\*/

public double getRangeUpperBound() {

return this.rangeMax.doubleValue();

}

/\*\*

\* Returns the upper bound for the range.

\* @param includeInterval include the y-interval?

\* @return The upper bound.

\*/

public double getRangeUpperBound(boolean includeInterval) {

return this.rangeMax.doubleValue();

}

/\*\*

\* Returns the range of values in the range (y-values).

\* @param includeInterval include the y-interval?

\* @return The range.

\*/

public Range getRangeBounds(boolean includeInterval) {

return this.range;

}

/\*\*

\* Returns the range of y-values.

\* @return The range.

\*/

public Range getValueRange() {

return this.range;

}

/\*\*

\* Returns the minimum domain value.

\* @return The minimum domain value.

\*/

public Number getMinimumDomainValue() {

return this.domainMin;

}

/\*\*

\* Returns the maximum domain value.

\* @return The maximum domain value.

\*/

public Number getMaximumDomainValue() {

return this.domainMax;

}

/\*\*

\* Returns the minimum range value.

\* @return The minimum range value.

\*/

public Number getMinimumRangeValue() {

return this.domainMin;

}

/\*\*

\* Returns the maximum range value.

\* @return The maximum range value.

\*/

public Number getMaximumRangeValue() {

return this.domainMax;

}

}