Corso di ingegneria informatica
A.A. 2021/2022

PRIMO PROGETTO

Nome dello studente:	*Nicola Carlucci*
Matricola	*554HHHINGINFOR*
Data Appello	*Dicembre 2019*
Esame	*Introduzione Big Data*

Indice generale

Premessa / Introduzione

L'elaborato in esame espone la risoluzione dei job e la spiegazione di come questi vengano analizzati e risolti seguendo il framework “MapReduce”

L'obiettivo di questa esercitazione è quello di prendere conoscenza e padronanza di come vengono implementate la fasi di 'Map' e la fasi di 'Reduce' su una mole arbitraria di dati che simula una minima quantità di 'Big dati' che vengono analizzati ogni secondo da complessi elaboratori.

Le operazioni di Map e di Reduce verranno implementate in codice java con l'ausilio della libreria Hadoop della Apache Software foundation (https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html).

I grafici verranno implementati con il software R Studio, quest'ultimo verrà utilizzato anche per confrontare i tempi di elaborazione del mapreduce con la libreria di hadoop implementata in java in un job di tipo wordcount.

Breve descrizione sul funzionamento teorico del MapReduce

Fase di Map

- I dati vengono suddivisi in split di dimensioni pari al blocco HDFS (128 Mb), ciascuno di questi viene inviato ad un blocco di elaborazione diverso(DataNode).
- Ciascuno split verrà sottoposto ad un map-task, il quale genererà l'output per il secondo approccio KeyVal(key,val).
- Gli output vengono poi distribuiti sui vari DataNode che fanno riferimento alla medesima operazione sui sottoinsiemi dei dati partizionati che gli sono arrivati in input.

Fase di Reduce

- Gli output dei singoli map-task vengono copiati in un singolo DataNode, il quale applica il reduce-task per elaborare il risultato finale.
- Se le risorse di un singolo nodo non sono sufficienti per l'analisi di tutti i map-task in un determinato intervallo di tempo, possono essere instanziati ulteriori nodi. Da ciò ne consegue che il il numero dei nodi non è legato al numero dei map-task ma dalle tempistiche che vogliamo ottenere.

Svolgimento

Sorgente dati in input.

I dati da prendere in esame per l'elaborazione dei job si presenteranno nel seguente modo :

2015-03-21,uova,latte,pane,vino

2015-05-18,pesce,pane,insalata,formaggio

.....

Il file contenente i dati è un semplice file di testo con estensione “txt”.

Il file che invece contiene i prezzi dei prodotti presenta la seguente struttura :

2015-5-28,uova/3,dolce/9,melanzane/2,formaggio/8,piadina/10,birra/4,parmigiano/13

2015-11-20,uova/3,melanzane/2,birra/4,piadina/10,pane/1,parmigiano/13

…...

Anche il file dei prezzi è un semplice file di testo con estensione “txt”.

Utility per lo sviluppo

LineMap.java

Ai fini di una buona programmazione e di una buona strutturazione del codice, può essere utile poter fare affidamento ad una classe che contenga i riferimenti ai prodotti, prezzi e numero di occorrenze.

La classe che ho deciso di creare prende il nome di “LineMap”, la sua struttura può essere vista nell’appendice (LineMap.java), come vedremo sarò molto utile per poter scrivere codice senza dover utilizzare un numero elevato di dizionari, raggruppare al meglio le informazioni relative ad una determinata data, ad una lista di prodotti, al prezzo totale …

Separator.java

Nella risoluzione del job ho incontrato numerose volte la necessità di inserire determinati caratteri, è buona norma non inserire mai delle black string nel codice, a tal fine ho deciso di inserire tutti i caratteri necessari per l'implementazione dei job in una classe denominata “Separator.java”

Job 1

Un job che sia in grado di generare, per ciascun mese del 2015, i cinque prodotti più venduti seguiti dal numero complessivo di pezzi venduti.

Per esempio:

2015-01: pane 852, latte 753, carne 544, vino 501, pesce 488

2015-02: latte 744, burro 655, uova 585, birra 498, pane 457

…....

Risoluzione del job

Da una prima analisi, si può evincere come può essere utile suddividere il job in due fasi:

1. Wordcount raggruppato per mensilità
2. Ordinamento e selezione dei primi cinque prodotti più venduti da una collezione ordinata.

Fase 1: Wordcount raggruppato per mensilità

Fase di Map

Il primo map-task consiste nel leggere l'input e strutturarlo per poterlo poi dare in input al keyval del mapreduce. La creazione dell'oggetto di tipo 'LineMap', dato un input di tipo stringa, può essere osservata dal metodo 'getLineMap(string,carattere di separazione)", presente nell'appendice nella sezione job1/part1/Map_Es_1.java.

Nel dettaglio:

- L'input viene sottoposta ad uno split array con il carattere ','
- Il primo elemento dell'array equivale alla data, poiché il giorno non interessa ai fini del job, viene eliminato => `//remove day to date`

```
String[] dateNoDayStringArray = fullDateString.split(Separator.DASH);
String dateNoDayString =
dateNoDayStringArray[0]+Separator.DASH+dateNoDayStringArray[1];
```

- Gli elementi restanti dello split array (fatta eccezione per la posizione 0 che corrisponde al valore della data che abbiamo appena acquisito) corrispondono ai prodotti dello scontrino.
- L'oggetto lineMap appena istanziato presenterà la struttura seguente:

  {

  dataString:'2015-01',

  value : ['acqua','birra',gelato...]

  }

Per poter personalizzare il map-task della libreria hadoop bisogna ricorrere all'override del metodo map(presente nell'appendice nella sezione job1/parte uno/Map_Es_1.java), il quale in seguito alla strutturazione del dato procede a richiamare il metodo keyVal(key,val), nel dettaglio:

- Per ogni prodotto appartenente alla lista dei prodotti del parametro 'value' dell'oggetto lineMap viene eseguito il keyval(key,val) dell'oggetto con i seguenti parametri :
    - key : la data con il relativo prodotto (Esempio 2015-01 birra)
    - value : il numero 1

Al termine della fase di Map avremo una struttura dati composta nel seguente modo :

2015-01 acqua, 1

2015-01 acqua, 1

2015-01 acqua, 1

2015-01 birra, 1

2015-01 birra, 1

….....

Fase di Reduce

Adesso possiamo iniziare ad eseguire la fase di reduce, per comodità anche questa parte la divideremo in ulteriori due parti:

1. Combiner: presente nell'appendice nella sezione job1/part1/Comb_Es_1.java, il quale si occupa di raggruppare gli elementi aventi la stessa chiave e incrementare il contatore somma con il valore intero che gli è stato assegnato precedentemente nel map-task.

Nel nostro caso il valore sarà sempre 1 perché stiamo effettuando un wordcount con costante 1.

Il combiner produrrà il seguente risultato :

2015-01 acqua, 35

2015-01 birra, 42

.....

2. Reduce : il reduce task, come si può osservare dall'override del metodo reduce presente nell'appendice nella sezione job1/part1/Reduce_Es_1.java, compone l'oggetto lineMap dalla riga di input in esame e la trasmette all'output con il seguente formato keyval (key, val):
    - key : la data dello scontrino
    - val : il prodotto con la relativa somma.

L'output del reduce-task sarà il seguente :

2015-01 ,acqua 35

2015-01 ,birra 42

.....

Fase 2, estrarre i primi 5 prodotti più venduti per ogni mese.

Fase di Map

La fase di map è cosi composta :

- Come prima operazione per poter eseguire nuovamente lo split dell'input tramite lo spazio vuoto, bisogna rimuovere i caratteri 'tab' presenti nell'output generato dal reduce-task precedente. Questa operazione è mostrata nel metodo 'removeSpaceWhiteLong()' , come si può vedere nell'appendice nella sezione job1/parte due/Map_Es1_part2.java.
- Creazione dell'oggetto lineMap tramite il metodo 'getLineMap(input,carattere di separazione)', l'input viene sottoposto ad uno split tramite il carattere 'spazio vuoto', il primo elemento dell'array corrisponde alla data dello scontrino(anno e mese), il secondo elemento indica il prodotto e il terzo elemento dell'array indica il numero di occorrenze totali del prodotto per quel determinato mese. Il metodo 'getLineMap()' è presente nell'appendice nella sezione job1/parte due/Map_Es1_part2.java.
- LineMap avrà le seguente struttura :

  { dataString: '2015-01' , value : 'birra 117' }
- Il map-task può essere implementato con l'override del metodo 'map' di Hadoop, come si può vedere nell'appendice nella sezione job1/parte due/Map_Es1_part2.java. Il map-task prosegue impostando il keyval(key,val) di map reduce nel seguente modo :
  - key : data dello scontrino
  - val : prodotto con la corrispondente somma mensile.

    (Esempio : 2015-01 , birra 33)

Fase di Reduce

In questo momento, i map-task hanno scritto una serie di output chiave,valore dove le chiavi rappresentano delle date raggruppate per mesi, e come valori ci sono i prodotti con la relativa somma.

Per poter indicare quali sono i cinque prodotti più venduti per ogni mese, ci occorrono due variabili globali di comodo:

```
List<LineMap> listLineMap = new ArrayList<>();

List<LineMap> resultLineMapList = new ArrayList<>();
```

1. Per ogni data, il reduce-task itera tutti i valori corrispondenti alla chiave in esame e li aggiunge alla listLineMap.
2. Il metodo getLineMapOrder si occupa di ordinare la lista, inserendo in cima l'elemento avente la somma maggiore. Il metodo può essere visto nell'appendice nella sezione job1/parte due/Reduce_Es1_part2.java.
3. Dalla lista ordinata vengono presi i primi 5 elementi e inseriti nella lista resultLineMapList, la quale contiene la lista definitiva dei valori da stampare come output del reduce-task.
4. L'output del reduce-task sarà composto nel seguente modo :

{ key : data , value : [birra : 290, acqua 230, limoni 190, gelato : 100, cioccolata : 50] }

L'override del metodo reduce può essere visto nel dettaglio nell'appendice nella sezione job1/parte due/Reduce_Es1_part2.java.

L'output del reduce-task produrrà il seguente risultato :

2015-1 insalata 34 pane 34 cioccolato 33 vino 32 acqua 31

2015-10 parmiggiano 35 cioccolato 31 uova 31 acqua 30 formaggio 30

2015-11 latte 29 mozzarella 29 parmiggiano 29 dolce 26 pesce 26

2015-12 latte 31 cioccolato 30 birra 29 melanzane 28 uova 28

2015-2 parmiggiano 39 pesce 34 piadina 34 uova 34 melanzane 33

2015-3 vino 33 mozzarella 32 pesce 32 pomodoro 32 piadina 31

2015-4 insalata 36 melanzane 35 uova 35 mozzarella 34 pesce 34

2015-5 pane 42 mozzarella 37 parmiggiano 37 melanzane 36 formaggio 35

2015-6 pesce 40 insalata 37 pomodoro 37 pane 36 piadina 36

2015-7 insalata 38 pesce 38 birra 36 dolce 36 pane 36

2015-8 acqua 38 mozzarella 36 parmiggiano 35 pesce 35 dolce 32

2015-9 uova 33 dolce 32 formaggio 32 melanzane 32 mozzarella 32

Per poter avere una visione globale che dia un'indicazione a colpo d'occhio di quelli che sono i risultati del mapreduce, possiamo utilizzare R-studio per la realizzazione di grafici.

L'output generato dal mapreduce è del tutto inutilizzabile per poter essere dato in input ad un elaboratore, a tal proposito il file è stato modificato con la seguente struttura csv:

Data,Prodotto,Tot

{val,val,val}

….....

R-studio dispone di molteplici package gratuiti che permettono la realizzazione di grafici con poche linee di codice, semplici ed intuitive. Il package che ho deciso di utilizzare è 'ggplot2', la documentazione ufficiale è disponibile al seguente link : https://www.r-graph-gallery.com/.

Come si può vedere dall'appendice nella sezione job1/parte due/job1.R, il codice produce il seguente risultato :

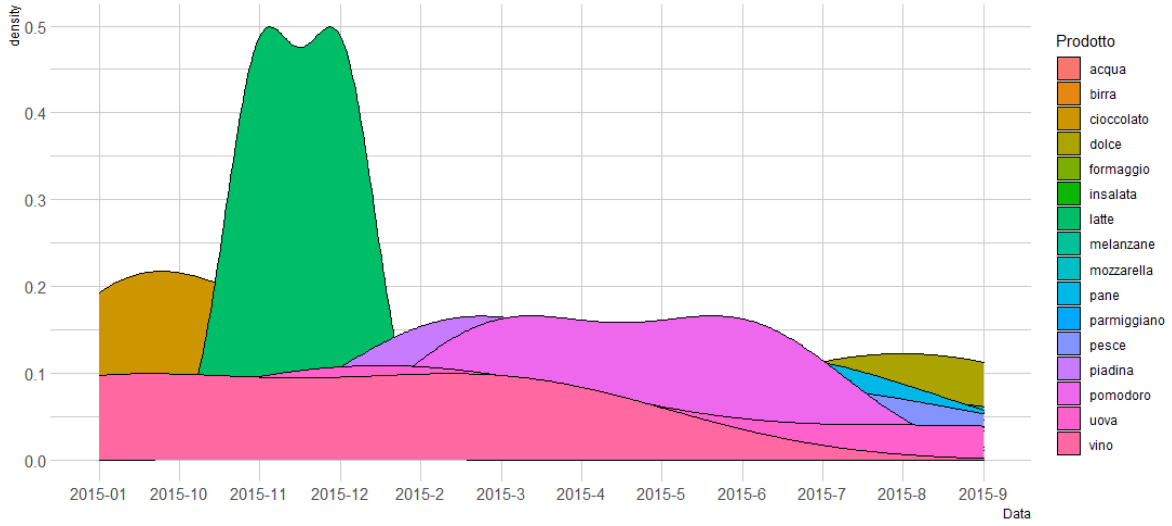

Illustrazione 1: Densità dei cinque prodotti più venduti per ogni mese(job1)

L'illustrazione 1 evidenzia la densità dei prodotti in un determinato intervallo di tempo, ad esempio è evidente come l'insalata appare con frequenza maggiore rispetto agli altri prodotti a Novembre e a Dicembre .

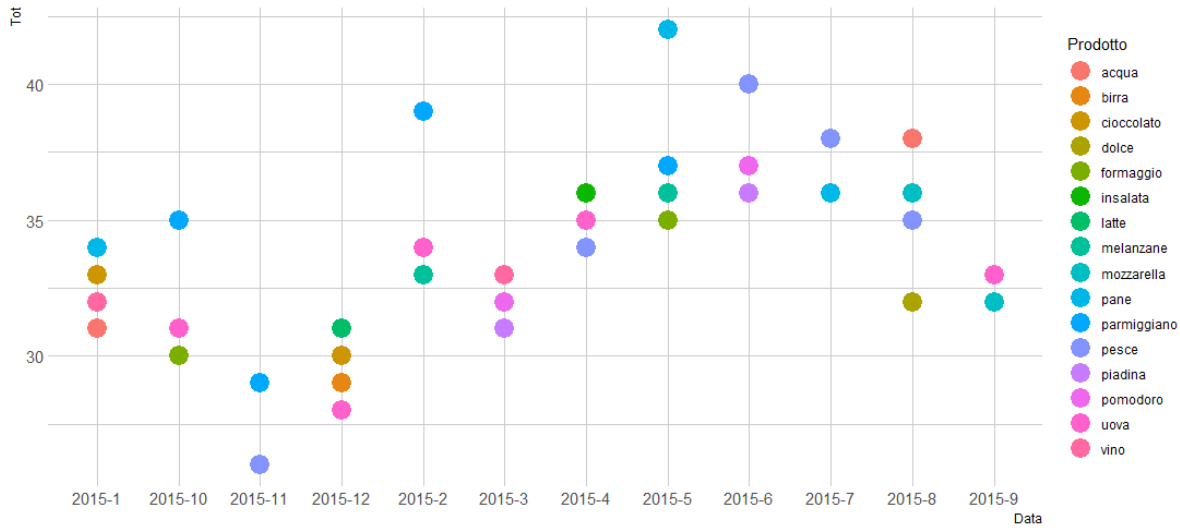

Illustrazione 2: Grafico a dispersione dei prodotti (job1)

L'illustrazione 2 evidenzia in una singola analisi le quantità dei prodotti venduti per ogni mese, ad esempio il pane è stato il prodotto con la maggior quantità di pezzi venduti non solo a Maggio 2015, ma rispetto a tutti i mesi.

Job 2

Un job che, dato un file in un formato a piacere contenente il costo di ciascun prodotto, sia in grado di generare, per ciascun prodotto, l'incasso totale per quel prodotto di ciascun mese del 2015.

Per esempio:

pane 1/2015:12340 2/2015:8530 3/2015:9450 …

latte 1/2015:11987 2/2015:10980 3/2015:12350 …

…

Risoluzione del job2

Come nella risoluzione del job1, anche il job2 può essere scomposto in 2 mini mapreduce per poter rendere più elastici i map-task.

- MapReduce 1 : calcola l'ammontare totale mensile di ogni prodotto
- MapReduce 2 : Raggruppa i prodotti e per ognuno di essi ne indica l'ammontare mensile.

MapReduce job2, parte uno

Fase di Map

Il map-task di questa fase è leggermente più articolato rispetto al job1, nel dettaglio :

- l'override del map-task, per ogni riga di input, instanzia una lista di lineMap object, la lista di lineMap viene passata all'override del metodo map tramite il metodo getListLineMap(input,carattere separatore), il suo codice è mostrato nell'appendice nella sezione job2/parte uno/Map_Es_2_part1.java.
- Il metodo getListLineMap, crea un array di stringhe derivante dallo split dell'input, come abbiamo visto precedentemente nel job1, il primo elemento dell'array corrisponde alla data, alla quale viene eliminato il giorno poiché non presente nell'output richiesto dal job2. In seguito, ogni valore restante dell'array viene passato in input ad un ulteriore metodo (getLineMap(input,carattere separatore)).
- Il metodo getLineMap(input,carattere separatore), come si può osservare nell'appendice nella sezione job2/parte uno/Map_Es_part1.java, non fa altro che analizzare l'input, fare uno split e far tornare un oggetto di tipo lineMap. Ad esempio :

  input : uova/3

  output : lineMap object [product : uova, intValue : 3]

- Giunti a questo punto, ogni valore dell'array è stato passato al metodo getLineMap, il quale si è occupato di creare un oggetto lineMap e far tornare l'oggetto al metodo getListLineMap che a sua volta, aggiungeva l'oggetto lineMap passato dal metodo getLineMap ad una lista di oggetti di tipo lineMap. Esempio di lista di lineMap che il metodo getListLineMap passerà come risultato all'override del metodo map :

  listLineMap<LineMap> : [

  dataString : '2015-05', product : 'uova', intValue: 3,

  dataString : '2015-05', product : 'dolce', intValue: 9,

  .....]

Questa è la lista di oggetti di tipo lineMap derivante dalla seguente riga di input :

2015-5-28,uova/3,dolce/9,melanzane/2,formaggio/8,piadina/10,birra/4,parmigiano/13

- Quando il metodo map riceve la lista di lineMap richiesta precedentemente al metodo getListLineMap, la itera, e per ogni oggetto presente in lista richiama la funzione keyval(key,val) nel seguente modo :
  - key : data e prodotto dell'oggetto lineMap
  - Val: valore intero associato

    Esempio: *2015-05 uova , 3*

Fase di Reduce

La fase di reduce è composta da due parti:

- Il combiner , come si può osservare nell'appendice nella sezione job2/parte uno/Comb_Es_2_part1.java si occupa di raggruppare tutti gli elementi aventi la stessa chiave (nel nostro caso la chiave sarà formata dalla coppia data-prodotto). Per ogni elemento avente la stessa chiave viene incrementato il contatore somma della quantità pari al valore dell'elemento associato alla chiave. Ad esempio
  - 2015-5 acqua , 10
  - 2015-5 acqua , 11
    - Output del combiner → 2015-5 acqua , 21

- Il reduce-task, come si può osservare nell'appendice nella sezione job2/parte uno/Reduce_Es_2_part1.java, si occupa di instanziare l'oggetto lineMap derivante dall'input per poterlo trasmettere al keyval(key,val).
  - Passa l'input come parametro al metodo getLineMap(input, carattere dello split), il quale compone l'oggetto lineMap e lo passa all'override del reduce-task.
  - Il reduce-task richiama il metodo keyval di hadoop con la seguente struttura :

  keyVal(data, prodotto e valore)

  Esempio :

  keyval(2015-05, acqua 35)

Il reduce-task produrrà il seguente risultato :

2015-1 acqua 62

2015-1 birra 120

2015-1 cioccolato 132

2015-1 dolce 252

2015-10 acqua 60

2015-10 birra 84

2015-10 cioccolato 124

….....

Il risultato del reduce-task verrà utilizzato nella fase 2 del mapreduce del job 2 come input.

Job2 Parte due

Fase di Map

- Il map-task, come si può osservare nell'appendice nella sezione job2/parte due/Map_Es_2_part2.java, viene implementato con l'override del metodo map.
- Il metodo map richiama il metodo getLineMap(input, carattere di separazione) passandogli come parametro l'input.
- Il metodo getLineMap, rimuove i caratteri 'tab' che sono stati precedentemente inseriti da hadoop della fase di reduce-task con il metodo removeTabSpace(string), il codice è osservabile nell'appendice nella sezione job2/parte due/Map_Es_2_part2.java.
- Una volta che i caratteri 'tab' sono stati rimossi e sostituiti con dei semplici 'spazi vuoti', si genera un array come risultato dello split dell'input. Il primo elemento dell'array corrisponde alla data, la quale stavolta necessita di cambiare forma rispetto alle soluzioni precedenti. Il metodo reformatDate(data, carattere di separazione) converte la data nel seguente modo:
  - Esempio : 2015-01 => 1/2015
- Gli elementi restanti dell'array corrispondo corrispettivamente al prodotto e al valore intero di esso. L'oggetto lineMap restituito all'override del metodo map presenterà la seguente struttura:
  - lineMap {dataString : String, product: String, intValue : int }

  Esempio getLineMap:

  Input → 2015-05 acqua 62

  output : lineMap { dataString : '2015-05', product: 'acqua', intValue:62 }
- Quando il metodo map riceve l'oggetto lineMap, richiama il metodo keyVal(key, val) con la seguente composizione:
  - key : product
  - val : data + ':' + int value

Esempio

keyval(acqua, 1/2015:62)

Fase di Reduce

- Il reduce-task si occupa di raggruppare tutti gli elementi aventi la stessa chiave, e per ognuno di essi copiare il valore corrispondente alla chiave in un unica variabile di testo.

Ad esempio :

acqua, 1/2015:62 , acqua, 2/2015:10 , acqua, 3/2015:8

==> acqua, 1/2015:62 2/2015:10 3/2015:8

L'override del reduce-task può essere visto nell'appendice nella sezione job2/parte due/Reduce_Es_2_part2.java.

L'output del reduce-task produrrà il seguente risultato:

acqua 1/2015:62 10/2015:60 11/2015:44 12/2015:52 2/2015:50 3/2015:54 4/2015:56 5/2015:56 6/2015:60 7/2015:70 8/2015:76 9/2015:50
birra 12/2015:116 3/2015:92 7/2015:144 2/2015:92 9/2015:104 8/2015:116 1/2015:120 6/2015:140 4/2015:132 10/2015:84 5/2015:132 11/2015:92
cioccolato 6/2015:136 4/2015:120 7/2015:124 3/2015:92 2/2015:112 8/2015:112 5/2015:132 9/2015:100 12/2015:120 1/2015:132 11/2015:100 10/2015:124
dolce 9/2015:288 12/2015:153 5/2015:288 10/2015:198 4/2015:297 8/2015:288 3/2015:234 11/2015:234 6/2015:297 2/2015:261 1/2015:252 7/2015:324
formaggio 6/2015:256 1/2015:200 11/2015:192 9/2015:256 12/2015:192 2/2015:224 3/2015:240 8/2015:208 4/2015:232 5/2015:280 10/2015:240 7/2015:256
insalata 1/2015:170 3/2015:130 10/2015:120 8/2015:135 2/2015:125 4/2015:180 7/2015:190 6/2015:185 9/2015:145 12/2015:95 5/2015:165 11/2015:110
latte 10/2015:156 8/2015:138 3/2015:168 12/2015:186 7/2015:192 6/2015:186 4/2015:192 1/2015:138 9/2015:156 5/2015:198 2/2015:180 11/2015:174
melanzane 9/2015:64 10/2015:56 4/2015:70 6/2015:70 11/2015:38 2/2015:66 5/2015:72 3/2015:46 12/2015:56 7/2015:66 1/2015:36 8/2015:56
mozzarella 1/2015:264 9/2015:352 11/2015:319 8/2015:396 3/2015:352 2/2015:330 10/2015:286 4/2015:374 7/2015:352 6/2015:374 5/2015:407 12/2015:297
pane 6/2015:36 5/2015:42 12/2015:21 2/2015:31 11/2015:23 3/2015:24 9/2015:29 10/2015:24 1/2015:34 4/2015:32 8/2015:30 7/2015:36
parmigiano 1/2015:351 7/2015:351 9/2015:377 3/2015:338 6/2015:416 2/2015:507 12/2015:286 5/2015:481 8/2015:455 11/2015:377 4/2015:416 10/2015:455

pesce 1/2015:264 10/2015:308 11/2015:286 12/2015:264 2/2015:374 3/2015:352 4/2015:374 9/2015:352 5/2015:352 6/2015:440 7/2015:418 8/2015:385
piadina 7/2015:290 11/2015:230 1/2015:220 2/2015:340 3/2015:310 4/2015:320 6/2015:360 5/2015:340 12/2015:270 8/2015:270 9/2015:290 10/2015:240
pomodoro 9/2015:81 6/2015:111 5/2015:90 4/2015:96 1/2015:90 7/2015:105 3/2015:96 2/2015:90 8/2015:87 12/2015:78 11/2015:75 10/2015:90
uova 2/2015:102 11/2015:78 6/2015:105 5/2015:93 4/2015:105 10/2015:93 7/2015:99 3/2015:84 12/2015:84 9/2015:99 1/2015:69 8/2015:78
vino 5/2015:476 7/2015:612 12/2015:425 1/2015:544 8/2015:476 9/2015:459 4/2015:425 10/2015:459 3/2015:561 6/2015:595 2/2015:561 11/2015:306

Anche in questo caso, l'output cosi come mostrato non può essere analizzato o dato in input ad un sistema di elaborazione, quindi ho strutturato l'output nel seguente modo :

Prodotto,Data,Tot

acqua,1/2015,62

acqua,10/2015,60

acqua,11/2015,44

…….....

L'output viene poi dato in input ad R-studio,il codice per la realizzazione del grafico è mostrato nell'appendice nella sezione job2/parte due/job2.R.

La libreria utilizzata per la realizzazione del grafico è 'ggplot2', la sua documentazione completa è disponibile al seguente link : https://www.r-graph-gallery.com.

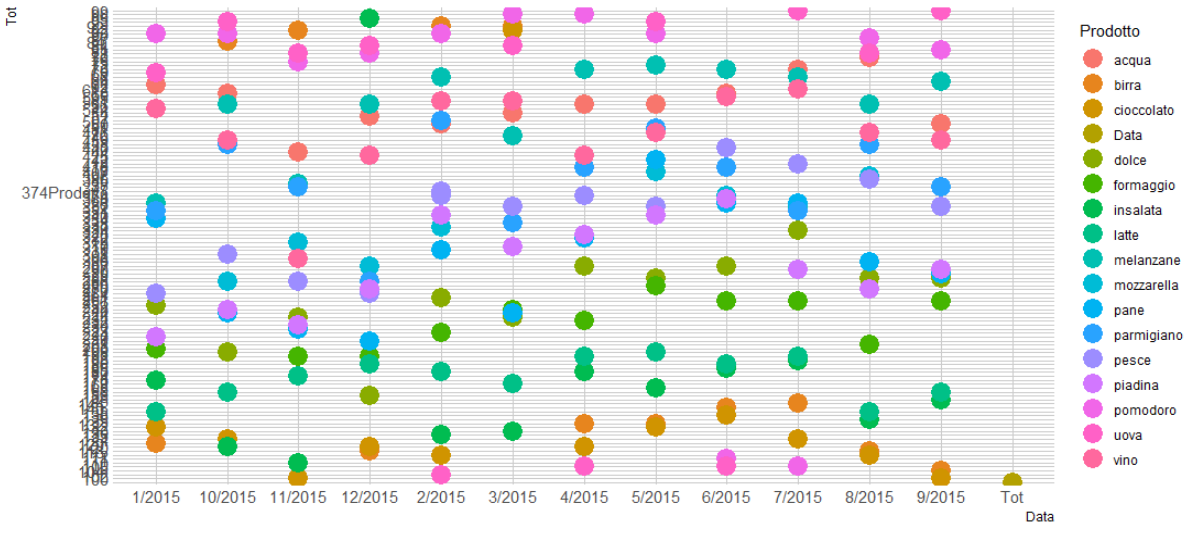

Illustrazione 3 : Distribuzione di frequenza del job2

Come si può osservare dall'illustrazione 3, si possono dedurre delle conclusioni in pochissimo tempo visionando il grafico con una certa attenzione:

- Il vino è uno dei prodotti che comporta un maggiore incasso, fatta eccezione per i mesi di Febbraio, Aprile e Giugno.
- Le melanzane sono uno dei prodotti più costanti con un incasso maggiore, cosi come l'acqua.
- La birra porta un incasso maggiore nei mesi di Ottobre, Novembre e Febbraio.
- ….....

job3

Un job in grado di generare, per ciascuna coppia di prodotti (p1,p2): (i) la percentuale del numero complessivo di scontrini nei quali i due prodotti compaiono insieme (supporto della regola di associazione p1→p2) e (ii) la percentuale del numero di scontrini che contengono p1 nei quali compare anche p2 (confidenza della regola di associazione p1→p2)

Per esempio :

pane,latte,30%, 4%

vino,uova,23%, 4%

latte,pane, 30%, 7%

Risoluzione del job3

La risoluzione del job3 richiede un'analisi molto più accurata e una risoluzione del mapreduce più dettagliata, andiamo a vedere nel dettaglio le fasi del mapreduce.

Fase di Map

I passi del map-task sono i seguenti :

- La data non è necessaria ai fini del task, l'input viene quindi passato al metodo removeDate(input, carattere di separazione) il quale si occupa di far ritornare l'input privato della data. Il metodo removeDate può essere visionato nell'appendice nella sezione job3/Map_Es_3.java.

Esempio :

  - input → 2015-05-28 uova,dolce, melanzane.
  - Output → uova dolce,melanzane

- Per poter confrontare tutte le possibili coppie, ho pensato di generare volta per volta tutte le possibili coppie attraverso la stringa di input. Il metodo che si occupa di generare tutte le possibili coppie è generateCouples(input) , visibile nell'appendice nella sezione job3/Map_Es_3.java. Prendendo come esempio l'input del punto precedente, il metodo produrrà il seguente output:

list {

uova dolce,

dolce uova,

uova melanzane,

melanzane uova,

dolce melanzane,

melanzane dolce

}

- Per ogni possibile coppia si verifica il supporto della regola di associazione e la confidenza della regola di associazione. Come primo passo si sottopone la stringa al metodo supportAssociation(input, coppia, carattere di separazione), il metodo può essere nell'appendice nella sezione job3/Map_Es_3.java.
- Il metodo supportAssociation (input, couple, carattere di separazione), verifica che i prodotti p1,p2 che formano la coppia 'couple' compaiono insieme, in caso affermativo il metodo fa tornare il seguente output:

  supportAssociation:1;confidentAssociation:0

  Nel caso in cui i prodotti p1,p2 non appaiono insieme nell'input, il metodo fa ritornare il valore 'null', questo perchè l'override del metodo map verifica l'output di 'supportAssociation', se l'ouptut non è nullo procede alla fase di keyval nel seguente modo:

  keyval(couple, supportAssociation : 1 ; confidentAssociation : 0 ; i : count)

  Soffermiamoci ad analizzare gli elementi dell'output:

  - couple → rappresenta la coppia formata da p1,p2
  - supportAssociation : 1 ; confidentAssociation : 0 → indica che la coppia è di supporto alla regola di associazione e quindi esclude la confidenza della regola di associazione
  - i: count → Per poter effettuare nel reduce-task i calcoli richiesti dal job, necessitiamo di conoscere il numero di input che sono stati analizzati, sicuramente sarebbe molto più semplice se il numero di input fosse stato letto inizialmente contando il numero di righe nel file, questo svolgimento però non avrebbe simulato un sistema reale nel quale i dati potrebbero arrivare in real-time. A tal proposito ho deciso di inserire una variabile di tipo contatore che si incrementa per ogni input analizzato, il valore del contatore viene poi inserito nel va dei keyVal(key,val) del map-task, in questo modo il reduce-task potrà effettuare dei calcoli efficienti e reali in real-time.

- Se il metodo 'supportAssociation' restituisce un output di valore null, viene verificata la regola di confidenza dell'associazione con il metodo confidentAssociation (input, couple, carattere di separazione) visibile nell'appendice nella sezione job3/Map_Es_3.java.
- Il metodo 'confidentAssociation' verifica che in un determinato input, siano entrambi presenti i prodotti p1,p2 che compongono la coppia 'couple', l'output del metodo è il seguente :

supportAssociation:0;confidentAssociation:0

supportAssociation:0;confidentAssociation:1

Il valore di confidentAssociation è '1' se la regola è verificata, '0' altrimenti.

- Il keyval del map-task viene chiamato nello stesso ed identico modo del supportAssociation, con la sola differenza che il val corrisponderà all'output del metodo confidentAssociation.
- Ogni qualvolta viene analizzato un nuovo input, come ho precedentemente spiegato, il metodo 'generateCouples' genera tutte le possibili coppie dell'input e le aggiunge ad una lista che contiene tutte le possibili coppie di tutti gli input : 'allCouples'. Questa lista ci torna utile perchè, una volta terminato di analizzare le regole di associazione dell'input corrente, dobbiamo aggiornare anche le regole di associazione dell'input corrente con le coppie precedentemente generate.
- Terminata l'analisi delle regole di associazione delle coppie generate dall'input corrente, si passa ad analizzare anche tutte le coppie che sono state generate in precedenza con l'input corrente, terminata quindi l'analisi di tutte le coppie, si passano le coppie generate dall'input corrente al metodo 'updateAllCouples(lista coppie)'. Come si può osservare nell'appendice nella sezione job3/Map_Es_3.java. , il metodo updateAllCouples si occupa di aggiornare la lista 'allCouples ' con le coppie che sono state generate dall'input corrente, se la coppia non è presente alla lista la aggiunge.

Fase di Reduce

La fase di Reduce del job 3 è composta da due parti :

1. Il combiner, come si può vedere nell'appendice nella sezione job3/Comb_Es_3.java, si occupa di raggruppare tutti gli elementi con le chiavi uguali, ed eseguire un sommatore del numero di occorrenze di 'supportAssociation' e di 'confidentAssociation'.

   La somma invece del numero di elementi funziona nel seguente modo:

   Se il count passato precedente dal map-task è inferiore alla somma corrente, non viene presa in considerazione, se invece il valore di count è superiore alla somma corrente, ne sovrascrive il valore. In questo modo dovremmo avere un output del combiner strutturato nel seguente modo :

   ```
   acqua,dolce;supportAssociation:36;confidentAssociation:50;i:90
   ```

2. Il reduce-task è implementato nell'override del metodo reduce come si può osservare nell'appendice nella sezione job3/Reduce_Es_3.java. L'input viene strutturato in un oggetto di tipo Association (vedere appendice Association.java) tramite il metodo 'getAssociation(input, carattere di separazione 1, carattere di separazione 2)'. L'oggetto di tipo Association avrà la seguente Struttura :

   Association {

   intTotal : “ ”,

   numSupportAssociation : “ ”,

   numConfidentAssociation : “ “

   }

   L'oggetto Association, affiancato all'input del reduce-task (in questo caso la coppia di prodotti) indica per ogni input le seguenti informazioni :

   - intTotal → il numero di input analizzati in totale

- numSupportAssociation → il numero di volte in cui è valida la regola di supporto della regola di associazione, e cioè il numero di volte in cui i prodotti p1,p2 compaiono insieme.
- numConfidentAssociation → il numero di volte in cui in input è presente p1, compare anche p2.

Infine, il reduce-task calcola il risultato finale per ogni coppia con le seguenti formule:

supporto della regola di associazione : (numSupportAssociation * 100) / intTotal

confidenza della regola di associazione : (numConfidentAssociation * 100) / intTotal

i risultati vengono passati in input al keyVal(key, val) del reduce-task nel seguente modo :

key → coppia

val → {supporto alla regola di associazione}% ,{confidenza della regola di associazione} %

Il risultato del reduce-task è il seguente :

acqua,birra 0.0% , 13.5%
acqua,cioccolato 0.0% , 14.2%
acqua,dolce 0.0% , 13.1%
acqua,formaggio 0.0% , 13.8%
acqua,insalata 0.0% , 13.9%
acqua,latte 9.6% , 3.5%
acqua,melanzane 0.0% , 13.1%
acqua,mozzarella 0.0% , 13.9%
acqua,pane 0.0% , 14.8%
acqua,parmiggiano 11.9% , 3.0%
acqua,pesce 0.0% , 14.2%
acqua,piadina 0.0% , 14.2%
acqua,pomodoro 0.0% , 15.5%
….....

L'output del mapreduce del job3 è composto da 240 linee di output che rappresentano tutte le possibili coppie del file di input. Realizzare un grafico con un numero cosi elevato di input non sarebbe ottimale e non soddisferebbe lo scopo di analizzare i risultati al fine di elaborare delle conclusioni, probabilmente si perderebbe più tempo a cercare di capire quale è l'input che stiamo cercando o analizzando nel grafico.

A tal proposito ho generato due input diversi che rappresentano rispettivamente :

1. Le prime coppie con la percentuale più elevata di supporto della regola di associazione.

Coppia,SupportoAssociazione,ConfidenzaAssociazione

acqua latte,9.6%,3.5%

pomodoro piadina,9.6%,6.0%

pomodoro pane,9.4%,5.2%

piadina acqua,9.2%,5.0%

latte parmiggiano,8.4%,8.4%

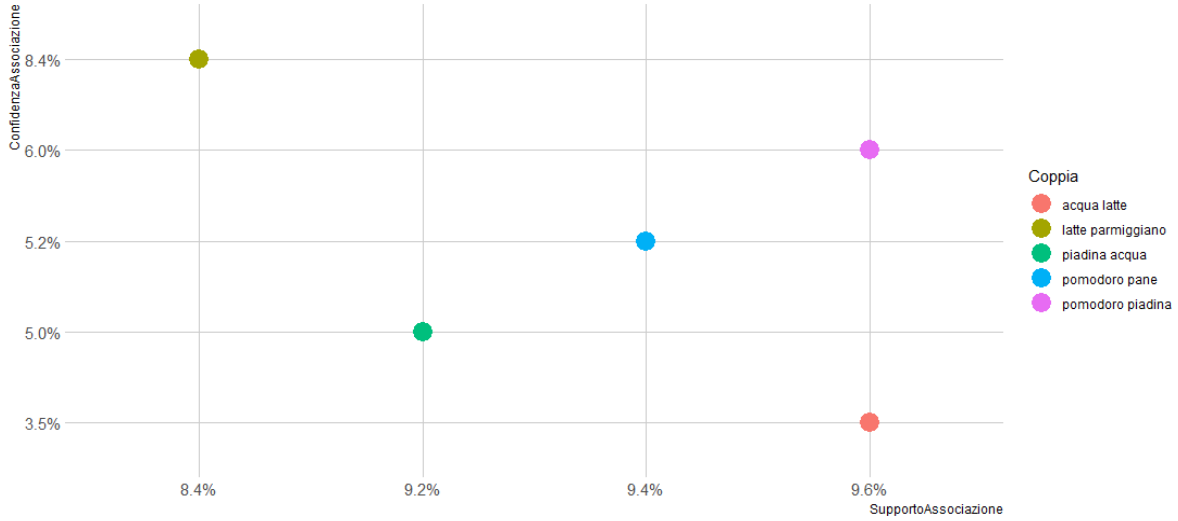

Illustrazione 4 : Le coppie con il più alto valore di supporto della regola di associazione

La coppia di prodotti con il più alto valore di supporto della regola di associazione è quella formata da (acqua,latte) e da (pomodoro,piadina), con la differenza che la coppia (pomodoro,piadina) ha un valore di confidenza della regola di associazione molto più elevato rispetto a quello di (acqua,latte).

La coppia (latte,parmiggiano) esprime lo stesso valore, quindi la probabilità che il latte sia acquistato insieme al parmiggiano è la stessa che in uno stesso scontrino ci sia sia il latte che il parmiggiano, e cioè dell'8.4%.

2. Le prime coppie con la percentuale più elevata di confidenza della regola di associazione.

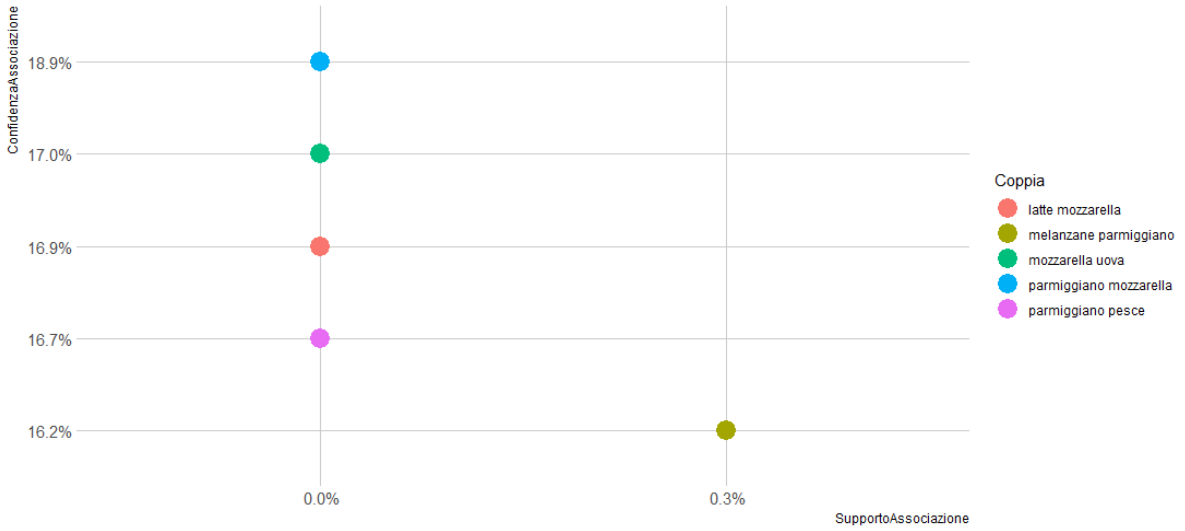

Illustrazione 5 : Le prime coppie con valore più elevato di confidenza della regola di associazione

Come si po' osservare dall'illustrazione 5, ci sono coppie di prodotti con un elevata percentuale di confidenza della regola di associazione, ma con una percentuale pari a zero di supporto della regola di associazione.

Osservando meglio il grafico si può osservare come le coppie con la probabilità più alta di

trovarsi nello stesso scontrino, hanno la probabilità più bassa in assoluto di essere comprati insieme.

Tempo di elaborazione

Può essere molto interessante confrontare i tempi di elaborazione di uno stesso job eseguito dalla libreria Hadoop in ambiente R ed in ambiente java.

Concettualmente le librerie producono lo stesso output e svolgono gli stessi task di map e di reduce, con la differenza che in java bisogna eseguire l'override dei metodi.

Per poter confrontare gli output eseguiti dalla libreria in ambienti di sviluppo diversi, ho preparato un job di tipo word-count che conta tutte le occorrenze dei prodotti del file di input del job1.

- Realizzazione in java

  Il codice del wordCount realizzato in java può essere visionato nell'appendice nella sezione Tempo di elaborazione/Java/Map_wordCount.java. Il codice è molto semplice, il map-task si occupa di impostare il metodo keyval(key,val) con la seguente struttura :

  key : prodotto

val : 1

Il reduce-task si occupa semplicemente di effettuare la somma di tutti i valori aventi la stessa chiave.

Il tempo di elaborazione del job è 1667 millisecondi.

- Realizzazione in R

  Ultimata l'installazione di Hadoop e di Rstudio, si procede all'avvio dei servizi con i seguenti comandi :

  start-dfs.sh

  start-yarn.sh

  hadoop fs -ls

  Si procede successivamente all'installazione dei pacchetti rmr2 e rhdfs con i seguenti comandi :

  install.packages("rhdfs")

  install.packages("rmr2")

  library(rhdfs)

  library(rmr2)

  hdfs.init()

  Il codice per la realizzazione del wordcount in R si può osservare nell'appendice nella sezione Tempo di elaborazione/R/WordCount.R.

  Il map-task è stato implementato nella funzione 'mapper', la quale si occupa di assegnare il valore 1 ad ogni chiave acquisita in input tramite il metodo keyval(key,val).

  Il reduce-task è implementato nel metodo 'reducerr'. Semplicemente conta la lunghezza delle occorrenze degli '1' assegnati ad ogni chiave ed effettua la somma.

  Tramite la funzione system.time(), R permette di mostrare il tempo di elaborazione delle istruzioni che gli sono state passate in input.

  Per calcolare il tempo di elaborazione del job ho inserito il mapreduce nella funzione 'timer',

  successivamente la funzione 'timer' viene passata come input al system.time, il risultato è il seguente :

User	System	Elapsed
9,268	1,800	10,418

  User indica il tempo utente che il sistema impiega per eseguire le istruzioni.

  System indica il tempo effettivo di elaborazione da parte del sistema.

  Elapsed indica il tempo di attesa per eseguire il risultato di una certa funzione.

Considerando che i due job sono stati eseguiti sulla stessa macchina fisica, a parità di memoria e di velocità di calcolo del processore, posso concludere affermando che i tempi di elaborazione sono minori se viene utilizzato l'ambiente java rispetto ad R.

Conclusioni

Successivamente alla realizzazione dei tre job, posso terminare questo elaborato con le seguenti conclusioni :

1. Il framework map-reduce è ottimale per analizzare, gestire ed elaborare grandi quantità di big-data.
2. Se gli output del reduce-task vengono analizzati attraverso l'implementazione di un grafico è molto più semplice poter osservare i comportamenti degli output e poterne trarre delle conclusioni velocemente e con una certa sicurezza.
3. É facile osservare come ci sono dei prodotti che ogni mese acquisiscono una certa importanza per i consumatori, a tal proposito si può usufruire di logiche aziendali volte a soddisfare le esigenze dei consumatori e dei venditori.
4. Ci sono prodotti che portano un maggiore incasso rispetto ad altri, il manager di un'azienda potrebbe analizzare questi dati e sfruttarli a proprio favore per poter migliorare ad esempio le strategie di vendita dei prodotti con incassi minori.
5. Analizzando le coppie di prodotti che sono venduti assieme, si potrebbe ad esempio posizionare gli elementi che compongono la coppia in scaffali vicini o in reparti comunicanti, in modo da rendere più gradevole l'esperienza dell'utente.

Utilizzare il MapReduce per l'implementazione dei job richiesti può quindi portare ad una mole di informazioni notevoli per migliorare le logiche e le strategie di business per un'azienda. Analizzando nel dettaglio magari un maggior numero di informazioni e di diverso carattere si possono trarre delle conclusioni che possono modificare quelle che sono le politiche aziendali, per poter apportare un guadagno superiore alla società che utilizza l'analisi dei Dati per il proprio business.

Appendice

LineMap.java

```
/**
 * Pojo utility for extraxct date and value to line
 *
 * @author Nicola Carlucci
 *
 */

public class LineMap {

	private String dataString;
	private List<String> value;
	private int intValue;
	private String product;

	public LineMap() {
		super();
	}

	public int getIntValue() {
		return intValue;
	}

	public void setIntValue(int intValue) {
		this.intValue = intValue;
	}

	public String getDataString() {
		return dataString;
	}

	public void setDataString(String dataString) {
		this.dataString = dataString;
	}

	public List<String> getValue() {
		return value;
	}

	public void setValue(List<String> value) {
		this.value = value;
	}

	public String getProduct() {
		return product;
	}

	public void setProduct(String product) {
		this.product = product;
	}
```

```
}
```

Separator.java

```
public class Separator {

	public static final String COMMA = ",";

	public static final String DASH = "-";

	public static final String SPACE = " ";

	public static final String DOUBLE_SPACE = "  ";

	public static final String SLASH = "/";

	public static final String TAB = "¥t";

	public static final String SEMICOLON = ";";

	public static final String DOUBLE_POINTS = ":";

	public static final String SUPPORT_ASSOCIATION = "supportAssociation:";

	public static final String CONFIDENT_ASSOCIATION = "confidentAssociation:";

}
```

Association.java

```
public class Association {

	private int total = 0;

	private Double numSupportAssociation = 0.0;

	private Double numConfidentAssociation = 0.0;


	public Association() {
	}


	public Association(int total, Double numSupportAssociation, Double
numConfidentAssociation) {
		super();
		this.total = total;
		this.numSupportAssociation = numSupportAssociation;
		this.numConfidentAssociation = numConfidentAssociation;
```

```
	}


	public int getTotal() {
		return total;
	}


	public void setTotal(int total) {
		this.total = total;
	}


	public Double getNumSupportAssociation() {
		return numSupportAssociation;
	}


	public void setNumSupportAssociation(Double numSupportAssociation) {
		this.numSupportAssociation = numSupportAssociation;
	}


	public Double getNumConfidentAssociation() {
		return numConfidentAssociation;
	}


	public void setNumConfidentAssociation(Double numConfidentAssociation) {
		this.numConfidentAssociation = numConfidentAssociation;
	}


	@Override
	public String toString() {
		return Separator.SUPPORT_ASSOCIATION + getNumSupportAssociation() +
Separator.SEMICOLON + Separator.CONFIDENT_ASSOCIATION + getNumConfidentAssociation() +
Separator.SEMICOLON + "i" + Separator.DOUBLE_POINTS + total;
	}



}
```

Job 1

Main_Es_1.java

```
/**
 * Main Job1
```

```
 *
 * @author Nicola Carlucci
 *
 */
public class Main_Es_1 extends Configured implements Tool {

	public static void main(String[] args) {

		int res_job1;
		try {
			System.out.println("Start job 1 --------->");
			res_job1 = ToolRunner.run(new Configuration(), new Main_Es_1(), args);
			System.exit(res_job1);
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			System.out.println("<----------------End job 1 --------->");
		}

	}

	@Override
	public int run(String[] arg0) throws Exception {
		Job job;
		Configuration configuration = new Configuration();
		String tempName = Utility.getMathRandom();
		try {
			job = new Job(configuration, "job1_part1");
			job.setJarByClass(Main_Es_1.class);

			job.setMapperClass(Map_Es_1.class);
			job.setCombinerClass(Comb_Es_1.class);
			job.setReducerClass(Reduce_Es_1.class);
			// math random name file

			System.out.println("File name is : Es_1_" + tempName + ".txt");

			FileInputFormat.addInputPath(job,
					new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/input/food.txt"));
			FileOutputFormat.setOutputPath(job,
					new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/output/Esercizio1/"
							+ tempName + "/Es_1_output_temp_" +
tempName + ".txt"));

			job.setOutputKeyClass(Text.class);
			job.setOutputValueClass(IntWritable.class);

			job.setInputFormatClass(TextInputFormat.class);
			job.setOutputFormatClass(TextOutputFormat.class);

			job.setMapOutputKeyClass(Text.class);
			job.setMapOutputValueClass(IntWritable.class);
```

```
                        if (job.waitForCompletion(true)) {
                                Job job2 = new Job(configuration, "job1_part2");

                                FileInputFormat.setInputPaths(job2,
                                                new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/output/Esercizio1/"
+ tempName
                                                                + "/Es_1_output_temp_" + tempName +
".txt"));
                                FileOutputFormat.setOutputPath(job2,
                                                new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/output/Esercizio1/"
                                                                + tempName + "/Es_1_output_" +
tempName + ".txt"));

                                // job2.setJarByClass(Job2Main.class);

                                job2.setMapperClass(Map_Es1_part2.class);
                                job2.setReducerClass(Reduce_Es1_part2.class);

                                job2.setInputFormatClass(TextInputFormat.class);
                                job2.setOutputFormatClass(TextOutputFormat.class);

                                job2.setMapOutputKeyClass(Text.class);
                                job2.setMapOutputValueClass(Text.class);

                                job2.setOutputKeyClass(Text.class);
                                job2.setOutputValueClass(Text.class);

                                job2.setNumReduceTasks(1);

                                if (job2.waitForCompletion(true)) {
                                        return 0;
                                } else {
                                        // error job 2
                                        System.out.println("error job 2");
                                        return -1;
                                }

                        } else {
                                // error job 1
                                System.out.println("error job 1");
                                return -1;
                        }

                } catch (Exception exception) {
                        exception.printStackTrace();
                        return -1;
                } finally {
                        System.out.println("File name is : Es_1_" + tempName + ".txt");
                }
        }
```

```
}
```

Parte uno

Map_Es_1.java

```
**
 * Map Esercizio 1 part 1
 *
 * @author Nicola Carlucci
 *
 */
public class Map_Es_1 extends Mapper<LongWritable, Text, Text, IntWritable> implements
UtilityLineMap{

	private static final IntWritable ONE = new IntWritable(1);

	@Override
	protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
IntWritable>.Context context)
			throws IOException, InterruptedException {

		LineMap lineMap = getLineMap(value.toString(),Separator.COMMA);
		if(lineMap!=null) {
			Text text = new Text();
			for(String product : lineMap.getValue()) {
				text.set(lineMap.getDataString()+" "+ product);
				context.write(text, ONE);

				// map job, insert in to context key, product 1
				// example---------------------->2015-01 birra 1
			}
		}
	}



	/**
	 * Create lineMap object from string
	 *
	 */
	@Override
	public LineMap getLineMap(String lineString, String separatorString) {
		LineMap lineMap = new LineMap();
		try {
			String[] arrayValue = lineString.split(separatorString);
			// date
			String fullDateString = arrayValue[0];

			//remove day to date
			String[] dateNoDayStringArray = fullDateString.split(Separator.DASH);
```

```
                        String dateNoDayString =
dateNoDayStringArray[0]+Separator.DASH+dateNoDayStringArray[1];

                        lineMap.setDataString(dateNoDayString);

                        //re-compose value
                        List<String> value = new ArrayList<String>();
                        for(int i=1;i<arrayValue.length;i++) {
                                value.add(arrayValue[i]);
                        }
                        lineMap.setValue(value);
                        // return data- list product
                        //example 2015-01 , List<acqua,birra,pane.....>
                }catch(Exception exception) {
                        exception.printStackTrace();
                        lineMap = null;
                }

                return lineMap;
        }

}
```

Comb_Es_1.java

```
/**
 * Combiner class job1 part 1
 *
 * @author Nicola Carlucci
 *
 */
public class Comb_Es_1 extends Reducer<Text, IntWritable, Text, IntWritable> {

        private final static IntWritable TOTAL = new IntWritable();

        @Override
        protected void reduce(Text arg0, Iterable<IntWritable> arg1,
                        Reducer<Text, IntWritable, Text, IntWritable>.Context arg2) throws
IOException, InterruptedException {

                int tot = 0;
                try {
                        for (IntWritable intWritable : arg1) {
                                tot = tot + intWritable.get();
                                // added 1 to element
```

```
                        // example i = 0 2015-01- acqua 1
                        // example i = 1 2015-01- acqua 2
                }
            } catch (Exception exception) {
                tot = -1;
            }
            Text text = new Text();
            text.set(arg0.toString());
            TOTAL.set(tot);
            arg2.write(text, TOTAL);
        }

}
```

Reduce_Es_1.java

```
/**
 * Reduce class job1 part1
 *
 * @author nikola
 *
 */
public class Reduce_Es_1 extends Reducer<Text, IntWritable, Text, Text> implements
UtilityLineMap {

        @Override
        protected void reduce(Text arg0, Iterable<IntWritable> arg1, Reducer<Text,
IntWritable, Text, Text>.Context arg2)
                        throws IOException, InterruptedException {

            int sum = 0;
            for (IntWritable intWritable : arg1) {
                sum = sum + intWritable.get();
            }
            LineMap lineMap = getLineMap(arg0.toString(), Separator.SPACE);
            if (lineMap != null) {
                Text textDate = new Text(lineMap.getDataString());
                Text textValue = new Text(lineMap.getValue().get(0) + Separator.SPACE +
sum);
                arg2.write(textDate, textValue);

                // 2015-01 acqua 31
                // 2015-01 birra 30
            }

        }

        /**
         * Create lineMap object from string
         *
         */
        @Override
        public LineMap getLineMap(String lineString, String separatorString) {
```

```
		LineMap lineMap = new LineMap();
		try {
			String[] values = lineString.split(separatorString);
			String date = values[0];
			String value = values[1];
			lineMap.setDataString(date);
			List<String> listValue = new ArrayList<>();
			listValue.add(value);
			lineMap.setValue(listValue);
		} catch (Exception exception) {
			exception.printStackTrace();
			lineMap = null;
		}
		return lineMap;
	}

}
```

Parte due

Map_Es1_part2.java

```
/**
 * Map class job1 part 2
 *
 * @author Nicola Carlucci
 *
 */
public class Map_Es1_part2 extends Mapper<LongWritable, Text, Text, Text> implements
UtilityLineMap {

	@Override
	protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
Text>.Context context)
			throws IOException, InterruptedException {

		LineMap lineMap = getLineMap(value.toString(), Separator.SPACE);
		Text dateText = new Text(lineMap.getDataString());
		Text valueText = new Text(lineMap.getValue().toString());
		context.write(dateText, valueText);

	}

	/**
	 * Get lineMap object
	 */
	@Override
	public LineMap getLineMap(String lineString, String separatorString) {
		LineMap lineMap = new LineMap();
		try {
			lineString = removeSpaceWhiteLong(lineString);
```

```
                        String[] lineStringArray = lineString.split(separatorString);
                        // set data
                        lineMap.setDataString(lineStringArray[0]);

                        // set value
                        List<String> stringList = new ArrayList<String>();
                        stringList.add(lineStringArray[1] + Separator.SPACE +
lineStringArray[2]);
                        lineMap.setValue(stringList);
                } catch (Exception exception) {
                        exception.printStackTrace();
                        lineMap = null;
                }
                return lineMap;
        }

        /**
         * Remove tab space from lineString
         *
         * @param lineString
         * @return
         */
        private String removeSpaceWhiteLong(String lineString) {
                String newLineString = "";
                if (lineString.contains("¥t")) {
                        newLineString = lineString.replace("¥t", " ");
                }
                return newLineString;
        }

}
```

Reduce_Es1_part2.java

```
/**
 * Reduce class job1 part 2
 *
 * @author Nicola Carlucci
 *
 */
public class Reduce_Es1_part2 extends Reducer<Object, Object, Object, Object> implements
UtilityLineMap {

        @Override
        protected void reduce(Object arg0, Iterable<Object> arg1, Reducer<Object, Object,
Object, Object>.Context arg2)
                        throws IOException, InterruptedException {

                List<LineMap> listLineMap = new ArrayList<>();

                List<LineMap> resultLineMapList = new ArrayList<>();
```

```
		for (Object object : arg1) {
			LineMap lineMap = getLineMap(object.toString(), Separator.SPACE);
			if (lineMap != null) {
				// add element to key
				// example 2015-01: pane 852
				// 2015-01: latte 753
				listLineMap.add(lineMap);
			}
		}

		// get first 5 elements
		if (listLineMap.size() != 0) {
			listLineMap = getLineMapOrder(listLineMap);
			resultLineMapList = listLineMap.subList(0, 5);
		}

		Text dateText = new Text(arg0.toString());
		String resultString = "";
		for (LineMap result : resultLineMapList) {
			// result
			// example
			// 2015-01: pane 852, latte 753, carne 544, vino 501, pesce 488
			resultString = resultString + result.getProduct() +  Separator.SPACE
+String.valueOf(result.getIntValue()) +  Separator.SPACE;
		}

		if (!resultString.equalsIgnoreCase("")) {
			Text valueText = new Text(resultString);
			arg2.write(dateText, valueText);
		}

	}

	@Override
	public LineMap getLineMap(String lineString, String separatorString) {
		LineMap lineMap = new LineMap();
		try {
			lineString = lineString.replace("[", "").replace("]", "");
			String[] allString = lineString.split(separatorString);
			lineMap.setProduct(allString[0]);
			lineMap.setIntValue(Integer.parseInt(allString[1]));
		} catch (Exception exception) {
			lineMap = null;
		}
		return lineMap;
	}

	/**
	 * Order list lineMap objects from intValue
	 *
	 * @param listLineMap list to order
	 * @return list lineMap order
	 */
	private List<LineMap> getLineMapOrder(List<LineMap> listLineMap) {
```

```
            return
listLineMap.stream().sorted(Comparator.comparing(LineMap::getIntValue).reversed())
                        .collect(Collectors.toList());
      }

}
```

job1.R

```
job1resultDensity = read.csv("C:/Users/Nikola/Documents/R/job1/job1result.csv")

p1 <- ggplot(data=job1resultDensity, aes(x=Data, group=Prodotto, fill=Prodotto)) +
  geom_density(adjust=1.5, alpha=.4) +
  theme_ipsum()

p1

job1resultPlot = read.csv("C:/Users/Nikola/Documents/R/job1/job1result.csv")


p2 <- ggplot(data=job1resultPlot, aes(x=Data, y = Tot, color = Prodotto)) +
  geom_point(size = 6)+
  theme_ipsum()

p2
```

job2

Main_Es_2.java

```
/**
 * Main job2
 *
 * @author Nicola Carlucci
 *
 */
public class Main_Es_2 extends Configured implements Tool {

      public static void main(String[] args) {

            int res_job2;
            try {
                  System.out.println("Start job 2 --------->");
                  res_job2 = ToolRunner.run(new Configuration(), new Main_Es_2(), args);
                  System.exit(res_job2);
            } catch (Exception e) {
                  e.printStackTrace();
            } finally {
                  System.out.println("<----------------End job 2 --------->");
            }
```

```
	}

	@Override
	public int run(String[] arg0) throws Exception {
		Job job;
		Configuration configuration = new Configuration();
		String tempName = Utility.getMathRandom();
		try {
			job = new Job(configuration, "job2_part1");
			job.setJarByClass(Main_Es_2.class);

			job.setMapperClass(Map_Es_2_part1.class);
			job.setCombinerClass(Comb_Es_2_part1.class);
			job.setReducerClass(Reduce_Es_2_part1.class);
			// math random name file

			System.out.println("File name is : Es_2_output_temp_" + tempName +
".txt");
			FileInputFormat.addInputPath(job,
					new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/input/price.txt"));
			FileOutputFormat.setOutputPath(job,
					new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/output/Esercizio2/"
							+ tempName + "/Es_2_output_temp_" +
tempName + ".txt"));

			job.setOutputKeyClass(Text.class);
			job.setOutputValueClass(IntWritable.class);

			job.setInputFormatClass(TextInputFormat.class);
			job.setOutputFormatClass(TextOutputFormat.class);

			job.setMapOutputKeyClass(Text.class);
			job.setMapOutputValueClass(IntWritable.class);
			System.out.println("File name is : Es_2output_temp_" + tempName +
".txt");

			if (job.waitForCompletion(true)) {
				Job job2 = new Job(configuration, "job2_part2");

				FileInputFormat.setInputPaths(job2,
						new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/output/Esercizio2/"
								+ tempName + "/Es_2_output_temp_" +
tempName + ".txt"));
				FileOutputFormat.setOutputPath(job2,
						new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/output/Esercizio2/"
								+ tempName + "/Es_2_output_" +
tempName + ".txt"));

				// job2.setJarByClass(Job2Main.class);
```

```
                        job2.setMapperClass(Map_Es_2_part2.class);
                        job2.setReducerClass(Reduce_Es_2_part2.class);

                        job2.setInputFormatClass(TextInputFormat.class);
                        job2.setOutputFormatClass(TextOutputFormat.class);

                        job2.setMapOutputKeyClass(Text.class);
                        job2.setMapOutputValueClass(Text.class);

                        job2.setOutputKeyClass(Text.class);
                        job2.setOutputValueClass(Text.class);

                        job2.setNumReduceTasks(1);

                        if (job2.waitForCompletion(true)) {
                                return 0;
                        } else {
                                // error job 2
                                System.out.println("error job 2");
                                return -1;
                        }

                } else {
                        // error job 1
                        System.out.println("error job 1");
                        return -1;
                }

        } catch (Exception exception) {
                exception.printStackTrace();
                return -1;
        } finally {
                System.out.println("File name is : Es_2_output_temp_" + tempName +
".txt");
        }
    }

}
```

Parte uno

Map_Es_2_part1.java

```
**
 * Map class job2 part 1
 *
 * @author Nicola Carlucci
 *
 */
```

```
public class Map_Es_2_part1 extends Mapper<LongWritable, Text, Text, IntWritable> implements
UtilityLineMap {

	@Override
	protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
IntWritable>.Context context)
			throws IOException, InterruptedException {

		List<LineMap> lineMapList = getListLineMap(value.toString(), Separator.COMMA);
		if (lineMapList != null && lineMapList.size() != 0) {
			for (LineMap lineMap : lineMapList) {
				Text textKey = new Text(lineMap.getDataString() +
Separator.SPACE + lineMap.getProduct());
				context.write(textKey, new IntWritable(lineMap.getIntValue()));
				// 2015-05 acqua ->32
				// 2015-05 birra ->64
			}
		}

	}

	/**
	 * Get list lineMap objects from lineString example
	 * 2015-5-28,uova/3,dolce/9,melanzane/2,formaggio/8 to 2015-05 -> (uova:3)
	 * (dolce:9)......
	 *
	 * @param lineString      line to format
	 * @param separatorString
	 * @return list lineMap
	 */
	private List<LineMap> getListLineMap(String lineString, String separatorString) {
		List<LineMap> lineMapList = new ArrayList<LineMap>();
		// split line , example :
		// 2015-5-
28,uova/3,dolce/9,melanzane/2,formaggio/8,piadina/10,birra/4,parmigiano/13
		String[] lineStringArray = lineString.split(separatorString);

		// remove day to date
		String[] dateNoDayStringArray = lineStringArray[0].split(Separator.DASH);
		String dateNoDayString = dateNoDayStringArray[0] + Separator.DASH +
dateNoDayStringArray[1];

		for (int i = 1; i < lineStringArray.length; i++) {
			// no forEach because not use lineStringArray[0], is date
			LineMap lineMap = getLineMap(lineStringArray[i], Separator.SLASH);
			if (lineMap != null) {
				lineMap.setDataString(dateNoDayString);
				lineMapList.add(lineMap);
			}
		}
		return lineMapList;

	}
```

```
	@Override
	public LineMap getLineMap(String lineString, String separatorString) {
		LineMap lineMap = new LineMap();
		try {
			String[] lineStringArray = lineString.split(separatorString);
			lineMap.setProduct(lineStringArray[0]);
			lineMap.setIntValue(Integer.parseInt(lineStringArray[1]));
		} catch (Exception exception) {
			exception.printStackTrace();
			lineMap = null;
		}
		return lineMap;
	}

}
```

Comb_Es_2_part1.java

```
/**
 * Combiner class job2 part 1
 *
 * @author Nicola Carlucci
 *
 */
public class Comb_Es_2_part1 extends Reducer<Text, IntWritable, Text,
IntWritable> {

	@Override
	protected void reduce(Text arg0, Iterable<IntWritable> arg1,
			Reducer<Text, IntWritable, Text, IntWritable>.Context arg2)
throws IOException, InterruptedException {

		int sum_prod = 0;
		try {
			for (IntWritable intWritable : arg1) {
				// add value int to key
				sum_prod = sum_prod + intWritable.get();

				// 2015-05 acqua -> total
			}
		} catch (Exception exception) {
			exception.printStackTrace();
			sum_prod = -1;
		}

		Text textKey = new Text(arg0.toString());
```

```
            arg2.write(textKey, new IntWritable(sum_prod));

      }
}
```

Reduce_Es_2_part1.java

```
/**
 * Reduce class job2 part 1
 *
 * @author nikola
 *
 */
public class Reduce_Es_2_part1 extends Reducer<Text, IntWritable, Text, Text>
implements UtilityLineMap {

      @Override
      protected void reduce(Text arg0, Iterable<IntWritable> arg1,
Reducer<Text, IntWritable, Text, Text>.Context arg2)
                  throws IOException, InterruptedException {

            // re-check sum
            int total_product = 0;
            for (IntWritable intWritable : arg1) {
                  total_product = total_product + intWritable.get();
            }
            LineMap lineMap = getLineMap(arg0.toString(), Separator.SPACE);
            if (lineMap != null) {
                  Text textKey = new Text(lineMap.getDataString());
                  Text textValue = new Text(lineMap.getProduct() +
Separator.SPACE + total_product);
                  arg2.write(textKey, textValue);
                  // 2015-05 -> acqua 35
            }

      }

      @Override
      public LineMap getLineMap(String lineString, String separatorString) {
            LineMap lineMap = new LineMap();
            try {
                  String[] lineStringArray =
lineString.split(separatorString);
                  lineMap.setDataString(lineStringArray[0]);
```

```
                lineMap.setProduct(lineStringArray[1]);
            } catch (Exception exception) {
                exception.printStackTrace();
                lineMap = null;
            }
            return lineMap;
        }
}
```

Parte due

Map_Es_2_part2.java

```
public class Map_Es_2_part2 extends Mapper<LongWritable, Text, Text, Text>
implements UtilityLineMap{


        @Override
        protected void map(LongWritable key, Text value, Mapper<LongWritable,
Text, Text, Text>.Context context)
                throws IOException, InterruptedException {

            LineMap lineMap = getLineMap(value.toString(),Separator.SPACE);
            if(lineMap!=null) {
                Text textProduct = new Text(lineMap.getProduct());
                Text textValue = new Text(lineMap.getDataString() +
Separator.DOUBLE_POINTS + String.valueOf(lineMap.getIntValue()));
                //example acqua 1/2015:62
                context.write(textProduct, textValue);
            }


        }

        @Override
        public LineMap getLineMap(String lineString, String separatorString) {
            LineMap lineMap = new LineMap();
            try {
                // 20015-01    acqua 62 -> 20015-01 acqua 62
                lineString = removeTabSpace(lineString);
```

```
                String[] lineStringArray =
lineString.split(separatorString);
                lineMap.setProduct(lineStringArray[1]);
                lineMap.setDataString(reformatDate(lineStringArray[0],
Separator.DASH));
                lineMap.setIntValue(Integer.parseInt(lineStringArray[2]));
            }catch(Exception exception) {
                exception.printStackTrace();
                lineMap = null;
            }
            return lineMap;
      }

      /**
       * Remove tab space from string and replace with space
       *
       * @param lineString is string from remove tab
       * @return string another tab space
       */
      private String removeTabSpace(String lineString) {
            String newLineString = "";
            String separatorTabString = Separator.TAB;
            if(lineString.contains(separatorTabString)) {
                newLineString = lineString.replace(separatorTabString,
Separator.SPACE);
            }
            return newLineString;
      }

      /**
       * reformat String from 2015-01 to 1/2015
       *
       * @param dateString
       * @param separatorString
       * @return
       * @throws Exception
       */
      private String reformatDate(String dateString, String separatorString)
throws Exception{
            String[] dateStringArray = dateString.split(separatorString);
            return dateStringArray[1]+ Separator.SLASH + dateStringArray[0];
      }
}
```

Reduce_Es_2_part2.java

```
public class Reduce_Es_2_part2 extends Reducer<Object, Object, Object,
Object> {


	@Override
	protected void reduce(Object arg0, Iterable<Object> arg1,
Reducer<Object, Object, Object, Object>.Context arg2)
			throws IOException, InterruptedException {
		LineMap lineMap = new LineMap();
		lineMap.setProduct(arg0.toString());
		List<String> value = new ArrayList<String>();
		for(Object object : arg1) {
			value.add(object.toString());
		}
		lineMap.setValue(value);

		if(lineMap.getValue().size()!=0) {
			String result="";
			for(String dateAndAmount : lineMap.getValue()) {
				result = result + Separator.SPACE + dateAndAmount;
			}
			arg2.write(new Text(lineMap.getProduct()), new
Text(result));
		}

	}

}
```

job2.R

```
job2resultPlot = read.csv("C:/Users/Nikola/Documents/R/job2/job2result.csv")

p2 <- ggplot(data=job2resultPlot, aes(x=Data, y = Tot, color = Prodotto)) +
  geom_point(size = 1)+ geom_rug()
  theme_ipsum()

p2
```

job 3

Main_Es_3.java

```
/**
 * Main job3
 *
 * @author Nicola Carlucci
 *
 */
public class Main_Es_3 extends Configured implements Tool {

	public static void main(String[] args) {

		int res_es3;
		try {
			System.out.println("Start job 3 --------->");
			res_es3 = ToolRunner.run(new Configuration(), new Main_Es_3(), args);
			System.exit(res_es3);
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			System.out.println("<----------------End job 3 --------->");
		}

	}

	@Override
	public int run(String[] arg0) throws Exception {
		Job job;
		Configuration configuration = new Configuration();
		String tempName = Utility.getMathRandom();
		try {
			job = new Job(configuration, "job3");
			job.setJarByClass(Main_Es_3.class);

			job.setMapperClass(Map_Es_3.class);
			job.setCombinerClass(Comb_Es_3.class);
			job.setReducerClass(Reduce_Es_3.class);
			// math random name file

			System.out.println("File name is : Es_3_" + tempName + ".txt");
			FileInputFormat.addInputPath(job,
					new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/input/food.txt"));
			FileOutputFormat.setOutputPath(job,
					new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/output/Esercizio3/"
							+ tempName + "/Es_3_" + tempName +
".txt"));

			job.setOutputKeyClass(Text.class);
			job.setOutputValueClass(IntWritable.class);
```

```
                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(Text.class);

                if (job.waitForCompletion(true)) {
                        return 0;
                } else {
                        // error job 1
                        System.out.println("error job 1");
                        return -1;
                }

        } catch (Exception exception) {
                exception.printStackTrace();
                return -1;
        } finally {
                System.out.println("File name is : Es_3_" + tempName + ".txt");
        }
    }
}
```

Map_Es_3.java

```
/**
 * Map job 3
 *
 * @author Nicola Carlucci
 *
 */
public class Map_Es_3 extends Mapper<LongWritable, Text, Text, Text> {

    /**
     * List all couples
     */
    List<String> allCouples = new ArrayList<String>();

    /**
     * User for summary row file
     */
    int count = 0;

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
Text>.Context context)
                throws IOException, InterruptedException {

        count++;
        String lineString = value.toString();
        // remove date
```

```
                String lineStringNoDate = removeDate(lineString, Separator.COMMA);
                if (lineStringNoDate != null && lineStringNoDate != "") {
                        // get all couples
                        List<String> listCoupleline = generateCouples(lineStringNoDate);
                        if (listCoupleline != null) {
                                for (String couple : listCoupleline) {
                                        String association = supportAssociation(lineStringNoDate,
couple, Separator.COMMA);
                                        if (association != null) {
                                                association = association + Separator.SEMICOLON +
"i:" + String.valueOf(count);
                                                context.write(new Text(couple), new
Text(association));
                                        } else {
                                                String confident =
confidentAssociation(lineStringNoDate, couple, Separator.COMMA);
                                                if (confident != null) {
                                                        confident = confident +
Separator.SEMICOLON + "i:" + String.valueOf(count);
                                                        context.write(new Text(couple), new
Text(confident));
                                                }
                                        }
                                } // for

                                // add rule to all couples if not exists in lineString
                                for (String coupleold : allCouples) {
                                        if (!listCoupleline.contains(coupleold)) {
                                                String association =
supportAssociation(lineStringNoDate, coupleold, Separator.COMMA);
                                                if (association != null) {
                                                        //
supportAssociation:1;confidentAssociation:0
                                                        // add i: total
                                                        association = association +
Separator.SEMICOLON + "i:" + String.valueOf(count);
                                                        context.write(new Text(coupleold), new
Text(association));
                                                } else {
                                                        String confident =
confidentAssociation(lineStringNoDate, coupleold, Separator.COMMA);
                                                        if (confident != null) {
                                                                confident = confident +
Separator.SEMICOLON + "i:" + String.valueOf(count);
                                                                context.write(new Text(coupleold),
new Text(confident));
                                                        }
                                                }
                                        }
                                } // for

                                // and update list all couples
                                updateAllCouples(listCoupleline);
                        }
```

```
		}

	}

	/**
	 * this metod return null if support Association rule is not present, else
	 * return row example : supportAssociation:1;confidentAssociation:0
	 *
	 * @param linString       string to check rule
	 * @param couple          couple to check rule
	 * @param separatorString
	 * @return null or supportAssociation:1;confidentAssociation:0
	 */
	private String supportAssociation(String linString, String couple, String
separatorString) {
		try {
			String coupleArray[] = couple.split(Separator.COMMA);
			String p1 = coupleArray[0];
			String p2 = coupleArray[1];
			String[] lineStringArray = linString.split(separatorString);
			boolean supportAssociation = false;
			for (int i = 0; i < lineStringArray.length; i++) {
				String p1_Line = lineStringArray[i];
				if (p1_Line.equalsIgnoreCase(p1)) {
					if (i < lineStringArray.length - 1) {
						String p2_line = lineStringArray[i + 1];
						if (p2_line.equalsIgnoreCase(p2)) {
							supportAssociation = true;
						}
					}
				}
			}
			if (supportAssociation) {
				String contextWrite = Separator.SUPPORT_ASSOCIATION + "1" +
Separator.SEMICOLON
						+ Separator.CONFIDENT_ASSOCIATION + "0";
				return contextWrite;
			} else {
				return null;
			}
		} catch (Exception exception) {
			return null;
		}
	}

	/**
	 * return null if do any exception, else string example:
	 * supportAssociation:0;confidentAssociation:0 or
	 * supportAssociation:0;confidentAssociation:1
	 *
	 * @param linString       string to check rule
	 * @param couple          couple to check rule
	 * @param separatorString
```

```
         * @return null or supportAssociation:0;confidentAssociation:0 or
         *         supportAssociation:0;confidentAssociation:1
         */
        private String confidentAssociation(String linString, String couple, String
separatorString) {
                try {
                        String coupleArray[] = couple.split(Separator.COMMA);
                        String p1 = coupleArray[0];
                        String p2 = coupleArray[1];
                        String[] lineStringArray = linString.split(separatorString);
                        int confidentAssociation = 0;
                        for (int i = 0; i < lineStringArray.length; i++) {
                                String p1_Line = lineStringArray[i];
                                if (p1_Line.equalsIgnoreCase(p1)) {
                                        for (int j = 0; j < lineStringArray.length; j++) {
                                                String p2_Line = lineStringArray[j];
                                                if (p2_Line.equalsIgnoreCase(p2)) {
                                                        confidentAssociation = 1;
                                                        break;
                                                }
                                        }
                                }
                        }

                        String contextWrite = Separator.SUPPORT_ASSOCIATION + "0" +
Separator.SEMICOLON
                                        + Separator.CONFIDENT_ASSOCIATION + confidentAssociation;
                        return contextWrite;

                } catch (Exception exception) {
                        return null;
                }
        }

        /**
         * Update list allCouples
         *
         * @param newListCouples new list of couples
         */
        private void updateAllCouples(List<String> newListCouples) {
                for (String newCouple : newListCouples) {
                        if (!allCouples.contains(newCouple)) {
                                allCouples.add(newCouple);
                        }
                }
        }

        /**
         * Generate all couples from lineString Example lineString a,b,c to (a,b) (a,c)
         * (b,c)
         *
         * @param lineString
         * @return List all couples
         */
```

```
	private List<String> generateCouples(String lineString) {
		List<String> couples = new ArrayList<String>();
		try {
			String[] lineStringArray = lineString.split(Separator.COMMA);
			// generate all couples
			for (int i = 0; i < lineStringArray.length - 1; i++) {
				for (int j = i + 1; j < lineStringArray.length; j++) {
					couples.add(lineStringArray[i] + Separator.COMMA +
lineStringArray[j]);
					couples.add(lineStringArray[j] + Separator.COMMA +
lineStringArray[i]);
				}
			}
		} catch (Exception exception) {
			exception.printStackTrace();
			couples = null;
		}
		return couples;
	}

	/**
	 * Remove date from string example 2015-5-28,uova,dolce,melanzane to uova,
	 * dolce, melanzana
	 *
	 * @param lineString      string to split
	 * @param separatorString char separator
	 * @return string no date
	 */
	private String removeDate(String lineString, String separatorString) {
		try {
			String[] lineStringArray = lineString.split(separatorString);
			String lineStringNoDate = "";
			for (int i = 1; i < lineStringArray.length; i++) {
				lineStringNoDate = lineStringNoDate + (i != 1 ? Separator.COMMA
: "") + lineStringArray[i];
			}
			return lineStringNoDate;
		} catch (Exception exception) {
			exception.printStackTrace();
			return null;
		}
	}

}
```

Comb_Es_3.java

```
/**
 * Combiner class job3
 *
```

```
 * @author Nicola Carlucci
 *
 */
public class Comb_Es_3 extends Reducer<Text, IntWritable, Text, Text> {

	@Override
	protected void reduce(Text arg0, Iterable<IntWritable> arg1, Reducer<Text,
IntWritable, Text, Text>.Context arg2)
			throws IOException, InterruptedException {

		Association associationFinal = new Association();
		double numSupport = 0;
		double numConfident = 0;

		// add total
		for (Object object : arg1) {
			Association association = getAssociation(object.toString(),
Separator.SEMICOLON, Separator.DOUBLE_POINTS);
			if (association != null) {
				// set total
				if (associationFinal.getTotal() < association.getTotal()) {
					associationFinal.setTotal(association.getTotal());
				}
				// increment support association
				numSupport = numSupport +
association.getNumSupportAssociation();

				// increment confident association
				numConfident = numConfident +
association.getNumConfidentAssociation();

			}
		}
		// write context
		arg2.write(arg0,
				new Text(Separator.SUPPORT_ASSOCIATION + numSupport +
Separator.SEMICOLON
					+ Separator.CONFIDENT_ASSOCIATION + numConfident
+ Separator.SEMICOLON + "i"
					+ Separator.DOUBLE_POINTS +
associationFinal.getTotal()));

	}

	/**
	 * Read lineString and get object type Association
	 *
	 * @param lineString               string to format Object
	 * @param separatorString          char separator properties
	 * @param separatorStringProperies char separator value properties
	 * @return Association or null
	 */
	private Association getAssociation(String lineString, String separatorString, String
separatorStringProperies) {
```

```
		Association association = new Association();
		try {
			String[] lineStringArray = lineString.split(separatorString);
			for (String properties : lineStringArray) {
				String[] propertiesArray =
properties.split(separatorStringProperies);
				String keyProperties = propertiesArray[0] +
Separator.DOUBLE_POINTS;
				switch (keyProperties) {
				case Separator.SUPPORT_ASSOCIATION:

	association.setNumSupportAssociation(Double.parseDouble(propertiesArray[1]));
					break;
				case Separator.CONFIDENT_ASSOCIATION:

	association.setNumConfidentAssociation(Double.parseDouble(propertiesArray[1]));
					break;
				case "i:":

	association.setTotal(Integer.parseInt(propertiesArray[1]));
				}
			}
		} catch (Exception exception) {
			association = null;
			exception.printStackTrace();
		}
		return association;
	}

}
```

Reducer_Es3.java

```
/**
 * Reduce Class job 3
 *
 * @author Nicola Carlucci
 *
 */
public class Reduce_Es_3 extends Reducer<Text, IntWritable, Text, Text> {

	@Override
	protected void reduce(Text arg0, Iterable<IntWritable> arg1, Reducer<Text,
IntWritable, Text, Text>.Context arg2)
			throws IOException, InterruptedException {

		Association associationFinal = new Association();
		for (Object object : arg1) {
			Association association = getAssociation(object.toString(),
Separator.SEMICOLON, Separator.DOUBLE_POINTS);
			if (association != null) {
				// set total
```

```
                        if (associationFinal.getTotal() < association.getTotal()) {
                                associationFinal.setTotal(association.getTotal());
                        }
                        // increment support association
                        associationFinal.setNumSupportAssociation(
                                        associationFinal.getNumSupportAssociation() +
association.getNumSupportAssociation());

                        // increment confident association
                        associationFinal.setNumConfidentAssociation(
                                        associationFinal.getNumConfidentAssociation() +
association.getNumConfidentAssociation());

                }
        }

        double percentualSupportAssociation =
(associationFinal.getNumSupportAssociation() * 100)
                        / associationFinal.getTotal();

        double percentualConfidentAssociation =
(associationFinal.getNumConfidentAssociation() * 100)
                        / associationFinal.getTotal();

        // write context
        arg2.write(arg0, new Text(String.valueOf(percentualSupportAssociation) + "%" +
Separator.SPACE + Separator.COMMA
                        + Separator.SPACE +
String.valueOf(percentualConfidentAssociation) + "%"));

    }

    /**
     * Read lineString and get object type Association
     *
     * @param lineString               string to format Object
     * @param separatorString          char separator properties
     * @param separatorStringProperies char separator value properties
     * @return Association or null
     */
    private Association getAssociation(String lineString, String separatorString, String
separatorStringProperies) {
        Association association = new Association();
        try {
                String[] lineStringArray = lineString.split(separatorString);
                for (String properties : lineStringArray) {
                        String[] propertiesArray =
properties.split(separatorStringProperies);
                        String keyProperties = propertiesArray[0] +
Separator.DOUBLE_POINTS;
                        switch (keyProperties) {
                        case Separator.SUPPORT_ASSOCIATION:

    association.setNumSupportAssociation(Double.parseDouble(propertiesArray[1]));
```

```
                                break;
                        case Separator.CONFIDENT_ASSOCIATION:

        association.setNumConfidentAssociation(Double.parseDouble(propertiesArray[1]));
                                break;
                        case "i:":

        association.setTotal(Integer.parseInt(propertiesArray[1]));
                                }
                        }
                } catch (Exception exception) {
                        association = null;
                        exception.printStackTrace();
                }
                return association;
        }

}
```

Tempo di elaborazione

Java

Map_wordCount.java

```
public class Map__wordCount extends Mapper<LongWritable, Text, Text, IntWritable>{

        private static final IntWritable ONE = new IntWritable(1);

        @Override
        protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
IntWritable>.Context context)
                        throws IOException, InterruptedException {

                context.write(new Text(value.toString()), ONE);
        }

}
```

Reduce_wordCount.java

```
public class Reduce__wordCount extends Reducer<Text, IntWritable, Text, Text>{

        @Override
```

```
	protected void reduce(Text arg0, Iterable<IntWritable> arg1, Reducer<Text,
IntWritable, Text, Text>.Context arg2)
			throws IOException, InterruptedException {

		int sum = 0;
		for (IntWritable intWritable : arg1) {
			sum = sum + intWritable.get();
		}
		arg2.write(new Text(arg0.toString()), new Text(String.valueOf(sum)));

	}

}
```

Main_wordCount.java

```
public class Main_wordCount extends Configured implements Tool{

	public static void main(String[] args) {

		int res_job1;
		try {
			Date startTime = new Date();
			System.out.println("Start job word count --------->");
			res_job1 = ToolRunner.run(new Configuration(), new Main_wordCount(),
args);
			Date endTime = new Date();
			long timeDone = (endTime.getTime() - startTime.getTime());
			System.out.println("Word count done in  ---------> " + timeDone +
"seconds");
			System.exit(res_job1);
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			System.out.println("<----------------End job word count --------->");
		}

	}

	@Override
	public int run(String[] arg0) throws Exception {
		Job job;
		Configuration configuration = new Configuration();
		String tempName = Utility.getMathRandom();
		try {
			job = new Job(configuration, "job word count");
			job.setJarByClass(Main_wordCount.class);

			job.setMapperClass(Map__wordCount.class);
			job.setReducerClass(Reduce__wordCount.class);
			// math random name file
```

```
                    System.out.println("File name is : wordcount_" + tempName + ".txt");

                    FileInputFormat.addInputPath(job,
                                    new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/input/wordcount.txt"
));
                    FileOutputFormat.setOutputPath(job,
                                    new
Path("/home/nikola/Scrivania/workspaceIntroBigData/IntroBigData/src/data/output/WordCount/"
                                                    + tempName + "/wordcount" + tempName +
".txt"));

                    job.setOutputKeyClass(Text.class);
                    job.setOutputValueClass(Text.class);

                    job.setInputFormatClass(TextInputFormat.class);
                    job.setOutputFormatClass(TextOutputFormat.class);

                    job.setMapOutputKeyClass(Text.class);
                    job.setMapOutputValueClass(IntWritable.class);

                    if (job.waitForCompletion(true)) {
                      return 0;

                    } else {
                          // error job
                          System.out.println("error job word count");
                          return -1;
                    }

              } catch (Exception exception) {
                    exception.printStackTrace();
                    return -1;
              } finally {
                    System.out.println("File name is : wordcount_ "+ tempName + ".txt");
              }
        }


}
```

R

WordCount.R

```
timer = function(){
  mapreduce_job1 = from.dfs(
    mapreduce(
      input = hdfs_input,
      map = mapper,
      reduce = reducerr
    ))
}

mapper = function(., v){
    keyval(v, 1)
  }

f = read.table("/home/hduser/R/prova/wordcount.txt")
hdfs_input = to.dfs(f)

reducerr = function(k, vv){
  keyval(k, length(vv))
}
  mapreduce_job = from.dfs(
    mapreduce(
      input = hdfs_input,
      map = mapper,
      reduce = reducerr
    ))

  system.time(timer())

result = as.data.frame(cbind(mapreduce_job$key, mapreduce_job$val))
r
e
s
u
```