Recommender system-HW4

103062318 蔡尚倫

1. Please explain the meaning of each JAVA file

input format:
<userID, movieID, grade, date-of-grade>

Main.java {
Construct whole Recommendation system architecture Here use solution 2
Preprocessing -> UVGen -> do{
IterationUSol2
IterationVSol2
++it
}while(it <3) //stop condition that can up to user's favor
}

Preprocessing.java {
This class contains the mapReduce job that is in chage of handeling the preprocessing
The preprocessing consists of formating the input data M such that
for each user, the score is normalised accross the movies he rated.
The reducer outputs for the set S of movies a user rated :
the sum of all normalised ratings for a user, as well as the
cardinality of S

Mapper:
<userID, movieID, grade, date-of-grade> -> <key,value> = <userID,<movieID, grade, date-of-grade>>

Reducer(MultipleOutputs):
<userID,<movieID, grade, date-of-grade>> ->
one output
Mean = sum of user's all rating grade/sum of movie that user rates(This is normalize)
<key,value> = <'M',<userID,movieID,grade-Mean>>

another output
<key,value> = <userID,<sum of user's all rating grade,sum of movie that user rates>>

After Call Run method
return Math.sqrt(sum/cardinality)
}

UVGen.java {
Construct data structure for UV and Compute initial values for U_0 and V_0

Constructor:fill the private variable
Path dest, double meanSquare, int dim, int movies, int users

generate:
Generate U and V matrices with Gaussian values with mean 0 and sd 0.25 as this yields better results

}

IterationUSol2.java {
This class implements the first mapReduce job of Solution 2 as described in the paper

Mapper:
two kinds of output
if key is 'M'
<key,value> = <userID,<'M',movieID, grade>>
else if key is 'U'
<key,value> = <userID,<'U',movieID, grade>>

Reducer:
The reducer task takes as input all tuples of a row of of U along with the tuples of the corresponding row in M.
It constructs the data structures and read V from local memory
It performs the algorithm as described in the paper

It outputs the corresponding new row of U

Setup->getVMatrix(Read V Matrix in HDFS and Store information in private double[][] V in Reducer class)
Main idea:M = U*V

lineCol = for one of U's row multiply Every columns of V

and then calculate
sum_j+=V[s][j]*(M_r[j]-lineCol)
sum_s+=V[s][j]*V[s][j]

and then
if this user has ratings
    U_r[s] value = sum_j/sum_s
else
    U_r[s] value = 0.0

after iterate length of DIMENSIONS times we can update all U_r[s]
and then write out
<key,value> = <'U',<userID,Index of each DIMENSIONS,Value of each DIMENSIONS >>

}

IterationVSol2.java (Algorithm same as IterationUSol2.java){
This class implements the first mapReduce job of Solution 2 as described in the paper

Mapper:
two kinds of output
if key is 'M'
    <key,value> = <userID,<'M',movieID, grade>>
else if key is 'V'
    <key,value> = <userID,<'V',movieID, grade>>

Reducer:

The reducer task takes as input all tuples of a column of of V along with the tuples
of the corresponding column in M.
It constructs the data structures and read U from local memory
It performs the algorithm as described in the paper
It outputs the corresponding new column of V
Setup->getUMatrix(Read U Matrix in HDFS and Store information in private double[][] U in Reducer class)
Main idea:M = U*V

lineCol = Every rows of U multiply for one of V's col

and then calculate
sum_i+=U[i][r]*(M_m[i]-lineCol)
sum_s+=U[i][r]*U[i][r]

and then
if this user has ratings
V_m[r] value = sum_i/sum_s
else
V_m[r] value = 0.0

after iterate length of DIMENSIONS times we can update all V_r[s]
and then write out
<key,value> = <'V',<Index of each DIMENSIONS,userID,Value of each DIMENSIONS >>

}

IterationU.java(Solution 1 concept is same as Solution 2 but need too much RAM to Run Tasks){

Mapper:
The map task read lines of U V and M files and emits chunks of rows of U along with their corresponding rows
of M to the reduces tasks. The Mapper also emits a duplicate of V to each reduce task.

The number of reduce tasks is 88*r_f, where r_f is the number of reduce task that a reducer will handle
We use round robin to distribute accross the key space

(nreducers*repFactor) for round robin parallel compute so value need to keep the userID in value
if key is 'M'
	<key,value> = <userID%(nreducers*repFactor),<'M',userID,movieID, grade>>
else if key is 'U'
	<key,value> = <userID%(nreducers*repFactor),<'U',userID,movieID, grade>>

else if key is 'V'
	for (int i=0;i<(nreducers*repFactor);++i)
		<key,value> = <i,<'V',userID,movieID, grade>>
Reducer:
The reducer task takes as input a chunk of rows of of U along with their corresponding rows in M. It also takes as input V. The reducer first parses the tuples and construct the data structures,
then performs the algorithm as described in the paper
and for each row of the chunk, it outputs the corresponding new row of U

Main idea:M = U*V

lineCol = for one of U's row multiply Every columns of V

and then calculate
sum_j+=V[s][j]*(vectM[j]-lineCol)
sum_s+=V[s][j]*V[s][j]

and then
if this user has ratings
	U_r[s] value = sum_j/sum_s
else
	U_r[s] value = 0.0

after iterate length of DIMENSIONS times we can update all U_r[s]
and then write out

<key,value> = <'U',<userID,Index of each DIMENSIONS,Value of each DIMENSIONS >>

}

IterationV.java(Solution 1 concept is same as Solution 2 but need too much RAM to Run Tasks){

Mapper:
The map task read lines of U V and M files, and emits chunks of columns of V along with their corresponding
columns of M to the reduces tasks. The Mapper also emits a duplicate of U to each reduce task.
The number of reduce tasks is 88*r_f, where r_f is the number of reduce task that a reducer will handle
We use round robin to distribute accross the key space

(nreducers*repFactor) for round robin parallel compute so value need to keep the userID in value
if key is 'M'
	<key,value> = <userID%(nreducers*repFactor),<'M',userID,movieID, grade>>
else if key is 'V'
	<key,value> = <userID%(nreducers*repFactor),<'V',userID,movieID, grade>>

else if key is 'U'
	for (int i=0;i<(nreducers*repFactor);++i)
		<key,value> = <i,<'U',userID,movieID, grade>>
Reducer:
The reducer task takes as input a chunk of cols of of V along with thier corresponding cols of M.
It also takes as input U. The reducer recieve a set of tuples so it first needs to construct the
data structures that will be used for computation. It then perform the algorithm as described in
the paper.

and for each column of the chunk, it outputs the corresponding new column of V
It also outputs the rmse for each of the cols along with the number of non blanks in M for each of thecols
.
Main idea:M = U*V

lineCol = Every rows of U multiply for one of V's col

and then calculate
sum_i+=U[i][r]*(vectM[i]-lineCol)
sum_s+=U[i][r]*U[i][r]

and then
if this user has ratings
V_m[r] value = sum_i/sum_s
else
V_m[r] value = 0.0

after iterate length of DIMENSIONS times we can update all V_r[s]
and then write out
<key,value> = <'V',<Index of each DIMENSIONS,userID,Value of each DIMENSIONS >>

}

2. Draw a flow chart of this given recommender system.(Main.java)