ADVANCED DATA MINING

ASSIGNMENT – I

A Report on Implementation of Incremental Association Rule Mining Using Canonical Trees in MapReduce Framework

Instructor-In-Charge :
Dr. Poonam Goyal,
CSIS Deppt,
BITS Pilani-Pilani Campus

Project team :

Apoorva Goyal	2013H112187P
Sumit Khanna	2010A7PS109P

ABSTRACT :

In the Assignment –I of Advanced Data Mining (BITS CS G520), we present an implementation of Canonical Trees (a.k.a CanTrees),a data-structure for generating frequent item-sets in Incremental Database. Inspired by its naive Implementation as proposed in [1], we build an application to cater to generate frequent item-sets constrained to the attributes pertaining to different sub-population of the customers' sample space. This is followed by an analysis on comparison of item-sets Association Rules generated in original dataset to those generated for different sub-populations when CanTrees Implemented in a distributed manner are run (on Hadoop 1.0.3) .

WALK-THROUGH :

1 # Canonical Trees – An Overview

Canonical Trees, better known as CanTrees, in general, is designed for incremental mining. The construction of the CanTree only requires one database scan. In our CanTree, items are arranged according to some canonical order, which can be determined by the user prior to the mining process or at runtime during the mining process. Specifically, items can be consistently arranged in lexicographic order or alphabetical order. CanTree construction follows the following two properties :-

Property 1 The ordering of items is unaffected by the changes in frequency caused by incremental updates.

Property 2 The frequency of a node in the CanTree is at least as high as the sum of frequencies of its children .

Once the CanTree is constructed, we can mine frequent patterns from the tree in a fashion similar to FP-growth. In other words, we employ a divide-and-conquer approach. We form projected databases by traversing the paths upwards only.

Since items are consistently arranged according to some canonical order , one can guarantee the inclusion of all frequent items using just upward traversals. There is no worry about possible omission or doubly-counting of items. Hence, for CanTrees, there is no need for having both upward and downward traver sals

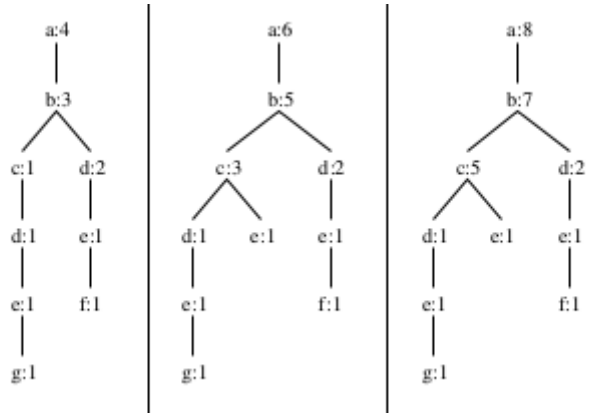
Following is an example for CanTree construction :-

The database from purchase log of the customers is shown below :-

		TID	Contents
DB	*Original DB*	t_1	$\{a, d, b, g, e, c\}$
		t_2	$\{d, f, b, a, e\}$
		t_3	$\{a\}$
		t_4	$\{d, a, b\}$
db_1	*1st group of insertions*	t_5	$\{a, c, b\}$
		t_6	$\{c, b, a, e\}$
db_2	*2nd group of insertions*	t_7	$\{a, b, c\}$
		t_8	$\{a, b, c\}$

Here, DB corresponds to the base dataset, and db1 and db2 are added as two incremental Transaction databases to it .

This is how a CanTree constructed out of DB and db1 and db2 looks like :-



Following the leafnodes, we can in a bottom-up fashion mine for the association rules on frequent item-sets / sub-sequences according to our constraints and thresholds (application-specific).

e.g :- for DB ,
The frequent item-sets are : d,b,a b,a b a

e.g for DB and db1 ,
The frequent item-sets are : c,b,a d,b,a c,b d,b a,b a b

e.g for DB and db1 and db2 ,
The frequent item-sets are : c,b,a d,b,a c,b d,b a,b a b

2 # **Our Implementation**

Data-set Generation/Pre-processing

1>We pick up the Census.dat – a dataset available on UCI -ML Repository . This data-set comprises of the tuples of the format :-

age,workclass,education,edu_num,marital,occupation,relationship,race,sex,gain,loss hours,country,salary

2> To every such tuple in the Census.dat file, we add a string of unique letters, that classifies as the items purchased by the customer.

For example,

AQWE age=middle-aged workclass=State-gov education=Bachelors edu_num=13 marital=Never-married occupation=Adm-clerical relationship=Not-in-family race=White sex=Male gain=medium loss=none hours=full-time country=United-States salary<=50K

means the customer with respective attributes purchases the items A,Q,W,E

For the sake of simplicity, we are keeping the size of Super-Market-Itemsets i.e |S| equal to 26, i.e the size of the English Alphabet. This however, is generic, and can be extended to any constant number R and applying a proper encoding schema to it.

3>Next, we take some tuples out of the modified Census.dat and enmark the following three datasets :

1 # The baseInput Dataset ~ DB
2 # The incr1Input Dataset ~ db1
3# The incr2Input Dataset ~ db2

Building CanTree on the Data-sets

Generating CanTrees for the Datta-sets is carried out in two fashions.

The first one comprises a simple sequential CanTree generation wherein we mine association rules out of the CanTrees that on the whole mines association rules for frequent item-sets in the original database DB and its incremental datsets db1 and db2

The second one comprises a similar run of CanTree but on DB firstly binned according to different attributes into different categories pertaining to different sub-population base . This comprises of two sub-sets of Mapreduce Tasks run in a mapreduce framework on Hadoop .

Sequential CanTree

This implementation of CanTrees works on the base dataset, generates a CanTree on it, mines the association rules for it and does the same with incremental datasets db1 and db2 added onto it.

This implementation only builds one CanTree for the entire dataset .After insertion of all the tuples from a particular dataset, it mines the association rules on it.

The description of CanTree generation is as follows:-

the driver class

the class is the driver class for the Sequential CanTree implementation. In the main method, specified are the paths where the input datasets are stored.

The CanTree class

The input datasets are processed in here,one by one,as follows :-

1>We take every string in the dataset, and sort the string according to alphabetic order. For simplicity, we take the alphabetic order as the metric for our CanTree.

2>We implement CanTree as a trie, wherein every node except the root node(of the CanTree) has the following attributes :-

- a char variable for the itemset
- an int to store the count of the itemset
- a pointer to the parent node
- an array of 26 pointers to child nodes

3>We perform the insertion of the strings as a normal tries datastructure insertion, alongwith the following checks :-

- increment the count of the item
- check if this the last char in the string, if yes then put it in the arraylist of leaf-nodes

4>After insertion of every itemstring gets over, we next follow a bottom-up approach to mine and print frequent itemsets occuring in the dataset. We query over every entry in the arraylist for leafnodes, and stop when count exceeds the thresholds for declaring an item as frequent. We then print all the ancestors to that node as a frequent item-set.

The results are stored seperately in the output files for DB, DB and db1, DB and db2.

CanTrees Implementation on Hadoop

-Prepare the base dataset, the incremental dataset 1, and the incremental dataset 2 out of the Census.dat modified above .

-We use the original attributes in Census.dat as the Meta-data for the customers, and the itemstrings added to it as the transaction data.

Workflow

We process te data through a series of two sub-sets of MapReduce tasks .

The first sub-set of MapReduce task aims at classifying the data above into various bins pertaining to different sub-population combinations .

The second sub-set runs in parallel CanTrees for every sub-population set of tuples and generates frequent itemsets for base sub-set as well as insertion of incremental datasets .

The set of tasks discussed above have been described below :-

Binning Task

The Binning Task comprises of Binning/Hashing jobs for each of the datasets. This task classifies the tuples in the datasets into 6 sub-population categories.

These 6 sub-population combinations are as follows :-

-”Middle-aged Black Male Poor”
-”Young Black Female Poor”
-”Young White Male Rich”
-”Middle-aged White Female Poor”
-Middle-aged Asian Female Poor'
-'Young Asian Male Rich”

The Binning Task Comprises of the following 3 jobs run in parallel :-

-BaseJob : This Job considers the various attributes of every tuple in the base dataset, and classifies the tuples into 6 categories as above.
-Incr1Job : This Job considers the various attributes of every tuple in the incremental dataset 1 and classifies the tuples into 6 categories as above.
-Incr2Job : This Job considers the various attributes of every tuple in the incremental dataset 2 and classifies the tuples into 6 categories as above.

The classes above have been explained below:-

BaseJob comprises of the BaseMapper and BaseReducer Functions. We are using the old API here, on Hadoop 1.0.3.
BaseMapper:-
Input key – LongWritable FileNumber
Inout Value – Text comprising the text in the base dataset

Output key – Text containing the name of the Sub-Population Category
Output Value – Text containing the itemstring corresponding to that tuple

The mapper basically considers attributes age,race,gender and income to classify every tuple into different categories.

BaseReducer:-
Input Key – Text containing the name of the Sub-Population Category
Input Value – Text containing the itemstring corresponding to that tuple

Output Key – Text containing the name of the Sub-Population Category
Input Value – An iterable list of the itemstrings in that category for base dataset

File: /user/hduser/work/baseDataOutput1_int1/part-00000

Goto : /user/hduser/work/baseDataOutp go

Go back to dir listing
Advanced view/download options

```
Middle-aged Asian Female Poor   YUINJ
Middle-aged Asian Female Poor   ROSHA
Middle-aged Black Male Poor     LIOPMNB
Middle-aged Black Male Poor     BAND
Middle-aged Black Male Poor     OPIUY
Middle-aged Black Male Poor     PINGE
Middle-aged White Female Poor   MNBVC
Middle-aged White Female Poor   SLACK
Middle-aged White Female Poor   GOYAL
Young Black Female Poor TRANQUIL
Young Black Female Poor ASDFG
Young Black Female Poor TIRED
```

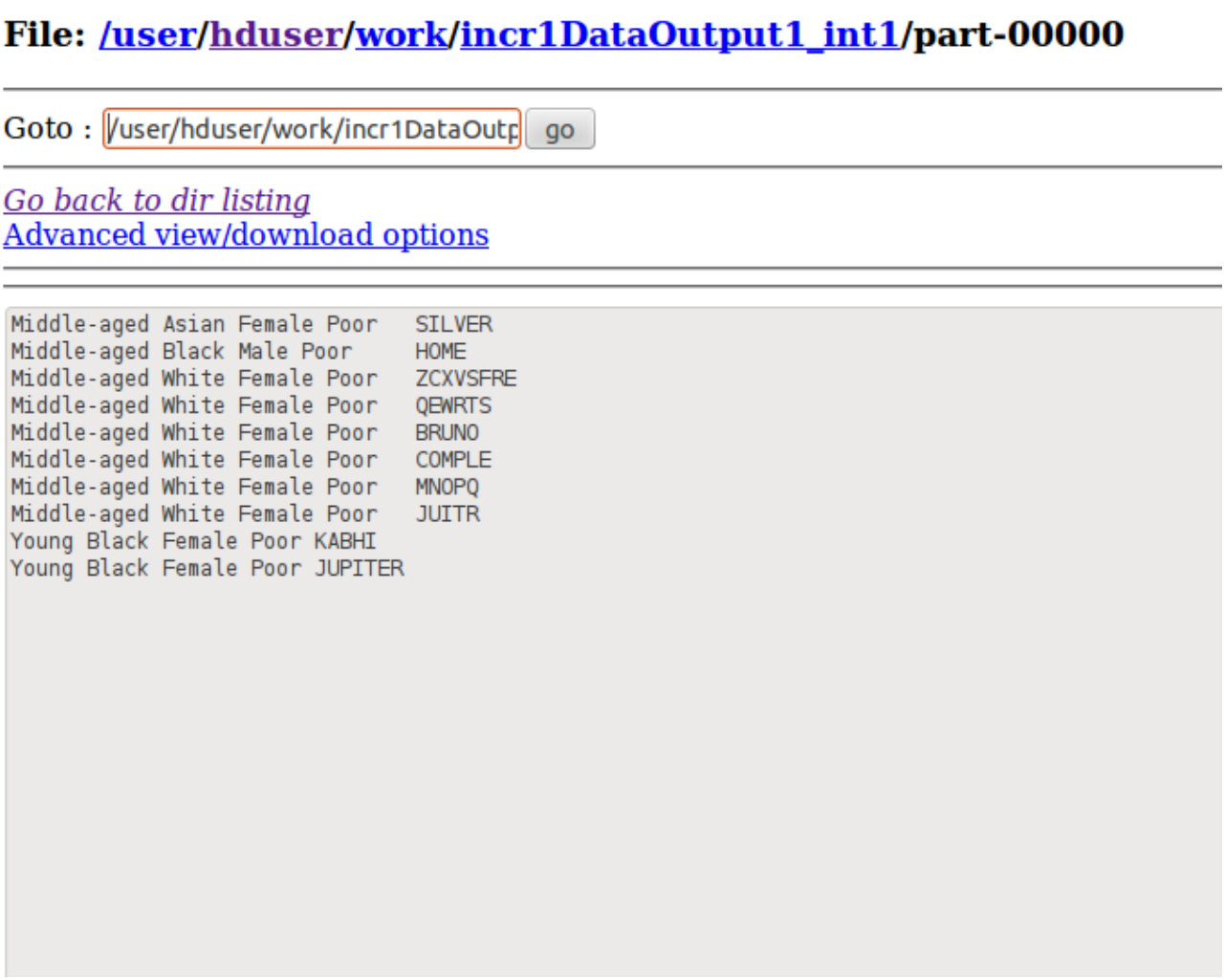
Incr1Job comprises of the Incr1Mapper and Incr1Reducer Functions. We are using the old API here, on Hadoop 1.0.3.
Incr1Mapper:-
Input key – LongWritable FileNumber
Inout Value – Text comprising the text in the base dataset

Output key – Text containing the name of the Sub-Population Category
Output Value – Text containing the itemstring corresponding to that tuple

The mapper basically considers attributes age,race,gender and income to classify every tuple into different categories.

Incr1Reducer:-
Input Key – Text containing the name of the Sub-Population Category
Input Value – Text containing the itemstring corresponding to that tuple

Output Key – Text containing the name of the Sub-Population Category
Input Value – An iterable list of the itemstrings in that category for incremental dataset 1



Incr2Job comprises of the Incr2Mapper and Incr2Reducer Functions. We are using the old API here, on Hadoop 1.0.3.
Incr2Mapper:-
Input key – LongWritable FileNumber
Inout Value – Text comprising the text in the base dataset

Output key – Text containing the name of the Sub-Population Category
Output Value – Text containing the itemstring corresponding to that tuple

The mapper basically considers attributes age,race,gender and income to classify every tuple into different categories.

Incr2Reducer:-
Input Key – Text containing the name of the Sub-Population Category

Input Value – Text containing the itemstring corresponding to that tuple

Output Key – Text containing the name of the Sub-Population Category
Input Value – An iterable list of the itemstrings in that category for incremental dataset 2

File: /user/hduser/work/incr2DataOutput1_int1/part-00000

Goto : /user/hduser/work/incr2DataOutp go

Go back to dir listing
Advanced view/download options

```
Middle-aged Asian Female Poor   PANICK
Middle-aged White Female Poor   PLUNGE
Middle-aged White Female Poor   SHUIO
Middle-aged White Female Poor   CLAY
Middle-aged White Female Poor   GHYUI
Middle-aged White Female Poor   SOCHI
Middle-aged White Female Poor   HELON
Middle-aged White Female Poor   DANCE
Young White Male Rich   PLAY
```

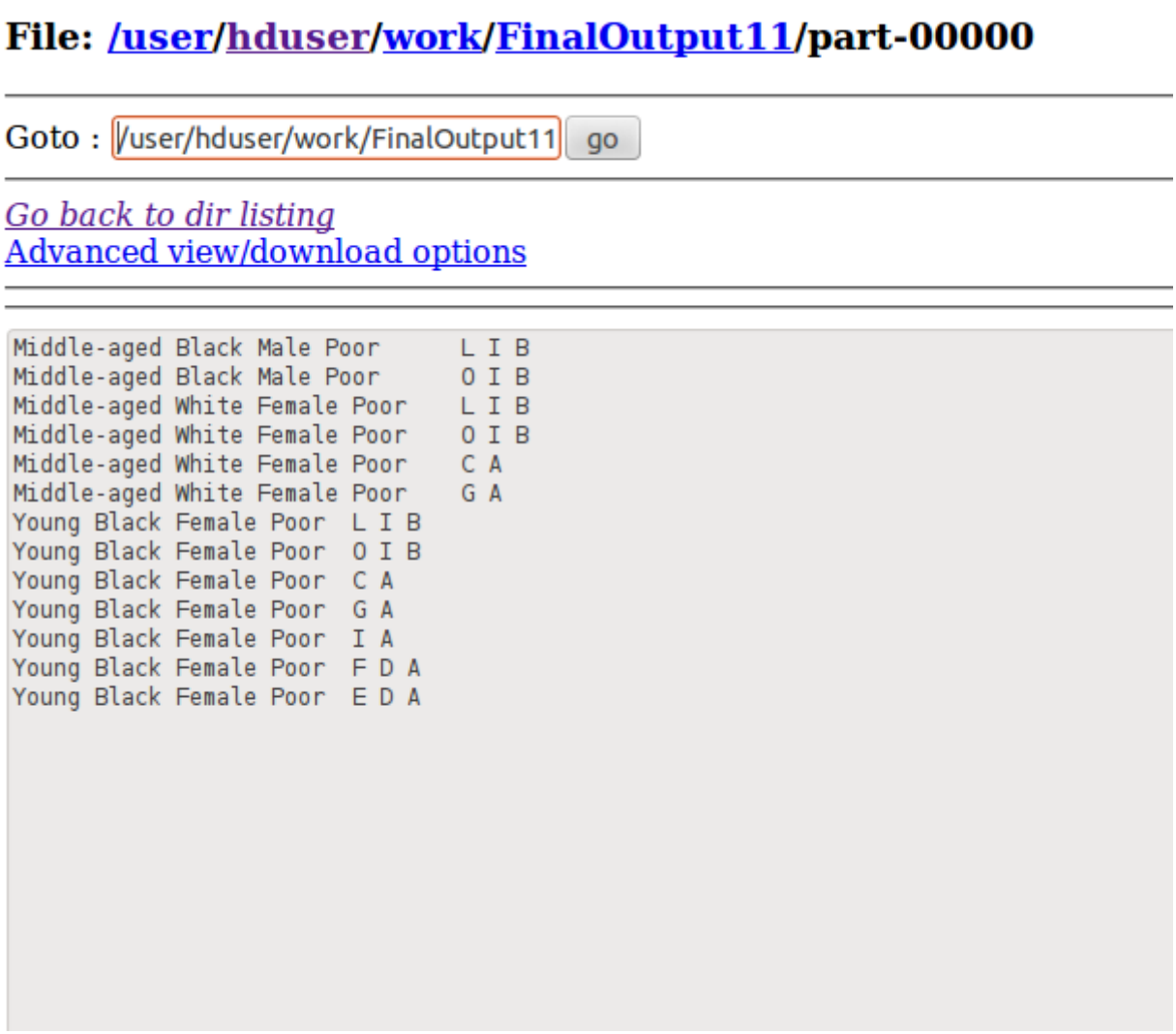
The output of these files is stored as an intermediate file/directory in the HDFS .

CanTrees Task

The CanTrees task comprises of a series of Jobs run on intermediate files corresponding to baseDataset and the two incremental datasets. Every task runs in parallel a CanTree for every sub-population. Further, the frequent itemsets are mined from the CanTrees generated, for every input datasets file.

CanTree task comprises of the following three Jobs:-

-BaseCanTreeJob : This Job constructs 6 CanTrees, One for each Sub-population Category and mines frequent itemsets for base dataset.



-Incr1CanTreeJob : This Job inserts into the CanTrees generated above, the itemstrings from incremental Dataset 1, and then mines the frequent itemsets affected by insertion of incremental dataset 1.

File: /user/hduser/work/FinalOutput21/part-00000

Goto : /user/hduser/work/FinalOutput21 go

Go back to dir listing
Advanced view/download options

```
Middle-aged White Female Poor    F E C
Middle-aged White Female Poor    Q E C
Middle-aged White Female Poor    O M L E C
Middle-aged White Female Poor    N M L E C
Young Black Female Poor  F E C
Young Black Female Poor  Q E C
Young Black Female Poor  O M L E C
Young Black Female Poor  N M L E C
```

-Incr2CanTreeJob : This Job inserts into the CanTrees generated above, the itemstrings from incremental Dataset 2, and then mines the frequent itemsets affected by insertion of incremental dataset 2.

File: /user/hduser/work/FinalOutput31/part-00000

Goto : /user/hduser/work/FinalOutput31 go

Go back to dir listing
Advanced view/download options

```
Middle-aged White Female Poor    L G E
Middle-aged White Female Poor    L C A
Middle-aged White Female Poor    H G E
Middle-aged White Female Poor    H C A
Middle-aged White Female Poor    H E
Middle-aged White Female Poor    C A
Young White Male Rich    L G E
Young White Male Rich    L C A
Young White Male Rich    H G E
Young White Male Rich    H C A
Young White Male Rich    H E
Young White Male Rich    C A
```

The implementation of the CanTrees is akin to the one discussed for the Sequential CanTrees.

Finally, the output is stored in different files for each dataset . The Output key corresponds to the Sub-Population Categories, and the Output Values correspond to the Frequent Itemsets discovered .

Contents of directory /user/hduser/work

Goto : /user/hduser/work go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
FinalOutput11	dir				2014-03-13 00:53	rwxr-xr-x	hduser	supergroup
FinalOutput21	dir				2014-03-13 00:54	rwxr-xr-x	hduser	supergroup
FinalOutput31	dir				2014-03-13 00:54	rwxr-xr-x	hduser	supergroup
baseDataInput	dir				2014-03-11 09:20	rwxr-xr-x	hduser	supergroup
baseDataInput1	dir				2014-03-13 00:49	rwxr-xr-x	hduser	supergroup
baseDataOutput1_int1	dir				2014-03-13 00:51	rwxr-xr-x	hduser	supergroup
incr1DataInput	dir				2014-03-11 09:23	rwxr-xr-x	hduser	supergroup
incr1DataInput1	dir				2014-03-13 00:49	rwxr-xr-x	hduser	supergroup
incr1DataOutput1_int1	dir				2014-03-13 00:52	rwxr-xr-x	hduser	supergroup
incr2DataInput	dir				2014-03-11 09:23	rwxr-xr-x	hduser	supergroup
incr2DataInput1	dir				2014-03-13 00:49	rwxr-xr-x	hduser	supergroup
incr2DataOutput1_int1	dir				2014-03-13 00:52	rwxr-xr-x	hduser	supergroup

Go back to DFS home

Local logs

Log directory

Results and Inference

The idea of this implementation is to come up with a better insight on the frequent item sets mined in an incremental data. Using the power of Hadoop framework for MapReduce, we can come up with Association Rules catering to the constraints of the Sub-Population Categories of the Customer Database. Although a similar concept may be applied sequentially as well, but with MapReduce paradigm shift, comes similar results in lesser time.

Some Itemsets may be frequent overall but may not pertain to a particular sub-category of the customers' population.