Draft 1 - Text Data Processing Techniques on Trouble Ticket for Enhancing Resolution Knowledge

Fauzy bin Che Yayah

November 21, 2015

*Abstract -Trouble ticket system is always includes very rich data, but how to data-mining from is very challenging task.In this paper , we applied the new technique of data processing for the ease for the next stage of classification and prediction.In other word , we also improvised the data mining technique by combining with the Bigdata technology for faster and achievable results.We realize that this techniques will provide luxuriant decision information for any organization or management and become a new fulfillment the field of data mining*

# Keywords

Data mining; data processing ; hadoop ; tickets data ; unstructured text processing

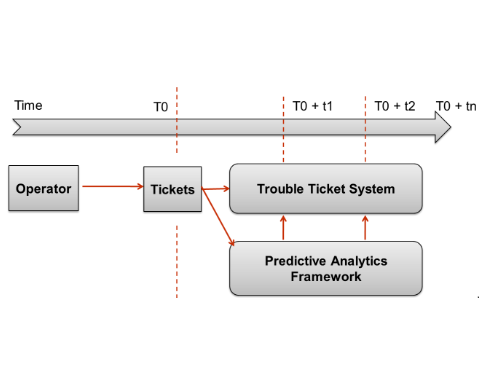
# Introduction

The trouble ticket system for telecommunication industries produced several thousand of records everyday. These records provide rich, accurate and complete for the system diagnostics and enhancing the customer experience by solving the problems. The live records stored inside the database of all levels of servers and replicated into the Enterprise Data Warehouse (EDWH) for data staging and shared among the departments via the credential and permission for different points of usage or use case. The data are coordinated and conserved for at least for five years minimum, which is enough for the analytics and data discovery purpose. The remaining more than five years of data will not be discarded and the archiving process will be applied for future reference. Because of the EDWH is optimized for storage, any heavily queries for analytics purpose in the system is not recommended which this activity will jeopardize the overall system performance.

# Related work

## Introduction of TTS

The trouble ticket system is considered as an assistant software which implements the Decision Support System (DSS) that manage and maintain lists of issues recorded by the customer support call center. The system contains a knowledge based containing the vital information for each subscribed customer, common resolution for most of the problems, and other such data. The ticket is the details of particular problem which contains the ticket status and other relevant data.



## Basic structure of trouble ticket data

The dataset of customer trouble ticket mainly comes from the trouble ticket system (TTS). The data set, then distributed into different levels of staging servers and other type database system at different levels of credential. The table below is a list of some fields from the trouble ticket table which mapped from the CTTS database :-

|  |  |  |
| --- | --- | --- |
| Column Name | Column Type | Notes |
| id | String | Generated ID by the system |
| created\_dt | DateTime | Date of the trouble ticket creation |
| closed\_dt | DateTime | Date of the trouble ticket closed |
| symp\_err | String | Symptom error detected |
| cause\_cat | String | Category Code define by the system |
| cause\_code | String | Cause Code define by the system |
| res\_code | String | Resolution Code define |
| desc | Text | Detail textual information such as |
|  |  | customer information, session status, |
|  |  | system status, network status , etc. |
|  |  | Cause Code define by the system |
| acc\_name | String | Name of the customer |
| acc\_num | String | Account number of the customer |
| package | String | Main package subscribed by the customers |
| zone | String | Product affected zone |
| .... | .... | .... |

In summary , this table below is the sample of TTS records :-

|  |  |  |
| --- | --- | --- |
| Column Type | Example | Field Name |
| Integer | [1001] , [1002] , [1003] | ID |
| DateTime | [2012-09-05 19:39:01.0],[2012-09-05 19:45:39.0] | created\_dt |
| Text | *"Troubleshooting Step: -check outages -check account status"* | desc |
| String | [Performance] , [Failure], [Open], [Closed] | symp\_err |
| .... | .... | .... |

We identify the structure inside the TTS (*desc*) contains a free text form inside the description field. This is in need for search and analyze inside it. The data need to transform into something that can have a better understanding and more effective representation. A TTS description is fill-up by capturing from the customer call center or by the technician from the two major sources such as email and subsequent call from the customer.Thus, most of the data is not explicitly structured, it is also highly noisy (e.g, inconsistent formatting) and also very heterogenous (e.g., multiple language, such as Bahasa Melayu or English, system generated data, specific telecommunication term , abbreviation , acronyms and jargons),making it hard to analyze and search information from the raw data. We introduced an approach for automatically search for specific knowledge or topics from the collection of the free from text section (desc) inside the TTS.

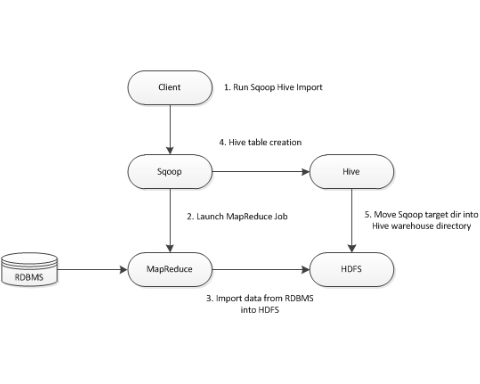
# Techniques

Our elegant methodology automatically structuring free-from heterogenous textual data helps to identify the structural pattern specific pattern known to the call center of technician which helps them to speed up the tedious work manually searching and understanding the relevant data inside it. Below is the workflow for this methodology :-

* Acquiring with basic filtering TTS dataset from EDWH into Hadoop using Sqoop
* Applying index number for TTS dataset inside Hive
* Second filtering of TTS dataset
* Create a Document Term Matrix (DTM)
* Sparse Terms Removal
* Vectorization of TTS (VTTS)

## Acquiring with basic filtering TTS dataset from EDWH into Hadoop using Sqoop

To acquire TTS dataset from the EDWH we need special operation using *Sqoop* components inside Hadoop into Hive compatible format. Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis.Sqoop is a tool designed to transfer data between Hadoop and relational database servers.



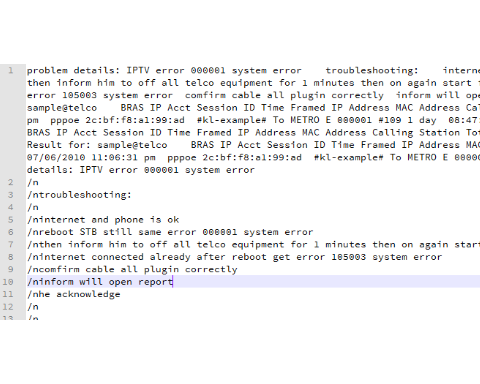
Sqoop basic syntax for importing dataset from TTS as follows :-

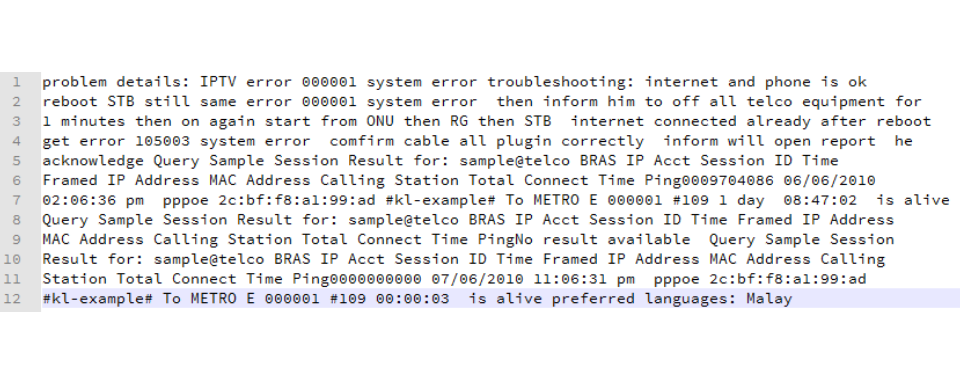
sqoop import --connect jdbc:[rdbms driver]:@[ip\_address]:[port]/[db\_name]   
--username [username] --password [password] --target-dir [temp\_target] --verbose   
-e "select [field[1:n],REGEXP\_REPLACE([desc\_field\_name],  
'[^a-zA-Z0-9\\:\\@\\/\\#\\-]', ' ') " --m 1 --hive-table   
[dbname\_in\_hdfs][tablename\_inhdfs] --hive-import  
--append --fields-terminated-by '\001'   
--lines-terminated-by '\n' --hive-delims-replacement '';

Description of the syntax above as follows :-

|  |  |
| --- | --- |
| Syntax | Description |
| sqoop import | Import a table from a database to HDFS |
| --connect jdbc:[rdbms driver] | Specify JDBC connect string |
| [ip\_address]:[port]/[db\_name] | Provides ip address , port no and rdbms database name |
| --username [username] | Provides the username |
| --password [password] | Provides the password |
| --target-dir [temp\_target] | Specify the target path for the output of the merge job |
| --verbose | Print information while execute |
| -e | specified query statement |
| *select* | statement returns a result set of records from one or more tables |
| [field[1:n] |  |
| REGEXP\_REPLACE([desc\_field\_name],) | search a string for a regular expression pattern |
| '[^a-zA-Z0-9\\:\\@\\/\\#\\-]', ' ') | remove all special characters and numbers |
| [-m 1] | Use n map tasks to import in parallel |
| [--hive-table | Sets the table name to use when importing to Hive |
| [[dbname\_in\_hdfs] | Sets the db name in Hive/HDFS |
| [tablename\_inhdfs] | Sets the db name in Hive/HDFS |
| --hive-import | Import tables into Hive format |
| --append | Append data to an existing dataset in HDFS |
| --fields-terminated-by '\001' | Sets the field separator character |
| --lines-terminated-by '\n' | Sets the end-of-line character |
| --hive-delims-replacement '' | Replace delimiter from string fields with user defined string |

To make, it is safe to resides into the Hive table, any special character set by REGEXP\_REPLACE found during acquiring the dataset will be replaced with default delimiter NULL or empty char. this can be achieved by applying the --hive-delims-replacement '' and REGEXP\_REPLACE [^a-zA-Z0-9\\:\\@\\/\\#\\-] inside the command line while sqoop is executed. The comparison images below shows the clean up results before and after applying the regular expression filtering :-





## Applying index number for TTS dataset inside Hive

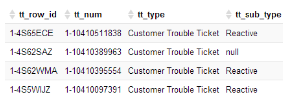
Basic default Hive dataset doesn't have any autonumber data type. The autonumber is very useful for primary key operation or for filter the records. We applied the the autonumber to the TTS dataset after the Sqoop operation finished. Because of the dataset which resides inside Hadoop is in Hive format , we need to use external jar file which is hive-contrib-0.12.0-cdh5.1.2.jar to create autonumber for the entire records. The jar file can be called by creating custom function as follows :-

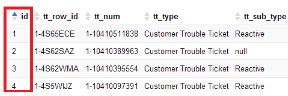
drop function IF EXISTS row\_sequence();   
create function IF NOT EXISTS row\_sequence()   
returns BIGINT location 'lib/hive-contrib-0.12.0-cdh5.1.2.jar'  
symbol='org.apache.hadoop.hive.contrib.udf.UDFRowSequence';

The rules applying the autonumber to the TTS dataset are to set by limits command inside the Hive-SQL. This is to ensure the dataset creation is reduced into single file inside the HDFS. Without the limit statement the Hive dataset files is chunked into the multiple split and the auto number sequence is set in 0 for each split or single reducer. This will cause the each records autonumber or it will be redundant. The Hive-SQL` command for this operation cab be achived by :-

drop table IF EXISTS tts\_stage1;   
create table IF NOT EXISTS tts\_stage1 as select row\_sequence() as id ,   
\* from [tts\_table] limit 10000;

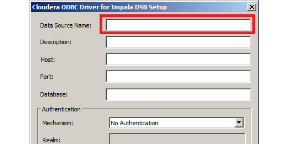
|  |  |
| --- | --- |
| SQL-Hive | Description |
| drop table IF EXISTS tts\_stage1; | Create table statement if exist |
| create table IF NOT EXISTS tts\_stage1 | Create table and add autonumber as the first field |
| as select row\_sequence() as id , | before the TTS fields |
| \* from [tts\_table] limit 10000; | Select all the fields from original TTS and limits for single reducer |





## Second filtering of TTS dataset

Since the dataset now are filled by autonumber id, now the record can be filtered by the id range . Tools such as RStudio which implements the R language statistical is used in this process. RStudio is capable to connect to various datasources including Hive. In this section , Open Database Connectivity(ODBC) for Hive need to be installed and configured. ODBC for Hive can be downloaded from the Cloudera website (<http://www.cloudera.com/content/www/en-us/downloads/connectors/hive/odbc/2-5-12.html>).



Set the ODBC data source name *[dsn\_name]* , the host address , the port number (default for *Impala* is 21050 ; *Hive* is 10000 ) , and the database name *[db\_name]*. After that , test the ODBC connection.

After the ODBC System DSN name have been set , use RStudio create new RScript and implements the R code below for connnecting to Hive ODBC via RODBC package. Package RODBC implements ODBC database connectivity. (<https://cran.r-project.org/web/packages/RODBC/vignettes/RODBC.pdf>).The Rstudio can be downloaded from the Rstudio website. (<https://www.rstudio.com/products/rstudio/download/>).

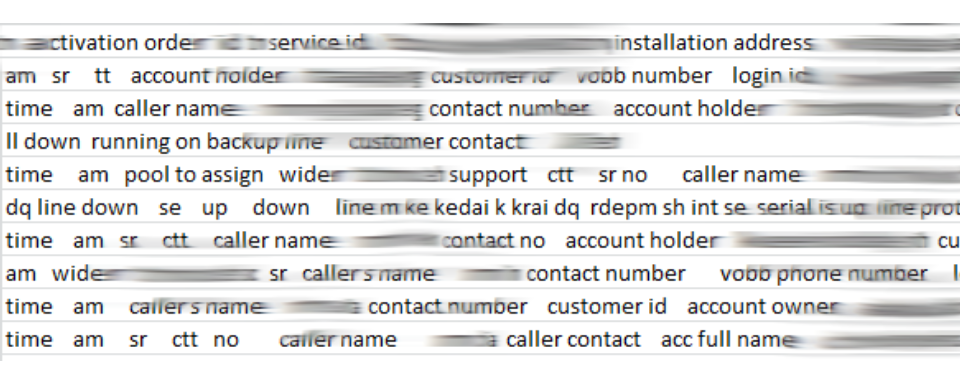
* Sample RScript to implements RODBC as follows :-
* library(RODBC)   
  conn <- odbcConnect(dsn="[dsn\_name]", uid="[username]", pwd="[username]") #  
  queryResult <- sqlQuery(conn, "select desc from [db\_name].[hive\_table\_with\_id]  
  where id > 0 and id <= 10000 ")  
  odbcClose(conn)

|  |  |
| --- | --- |
| R Script | Description |
| library(RODBC) | load the RODBC instance components |
| conn <- odbcConnect(dsn="[dsn\_name]"), | Set the DSN name |
| uid="[username]", pwd="[username]" | set username and password |
| sqlQuery(conn, "select desc from [hive\_table\_with\_id]" | Hive-SQL to filter the records |
| "where id > 0 and id <= 10000" | Condition to select the records |
| odbcClose(conn) | Cleans up and frees resources |

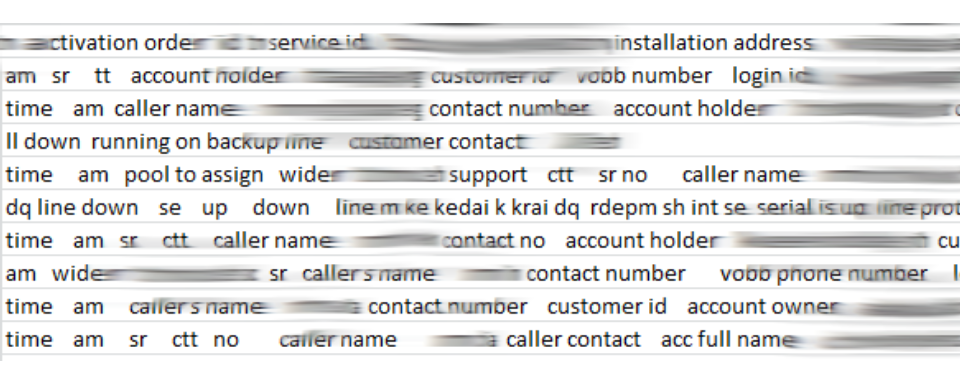
Five stages of data filteration need to be applied to clean up description text from the TTS information

|  |  |  |
| --- | --- | --- |
| # | Description | R Script with Regular Expression filter |
| 1 | Lowercase all the charactors | df\_desc <- as.data.frame(tolower(df\_desc$review)) |
| 2 | Remove all the numbers | df\_desc <-as.data.frame |
|  |  | (sapply(df\_desc,gsub,pattern="[[:digit:]]",replacement="")) |
| 3 | Remove all the punctuation | df\_desc<-as.data.frame |
|  | and symbols | (sapply(df\_desc,gsub,pattern="[[:punct:]]",replacement="")) |
| 4 | Trim the unnecessary spaces | df\_desc <-as.data.frame |
|  |  | (sapply(df\_desc,gsub,pattern="@\\w+",replacement="")) |
| 5 | Trimming the whitespace | df\_desc <-as.data.frame |
|  |  | (sapply(df\_desc,gsub,pattern="^\\s+|\\s+$",replacement="")) |

The image below shows state of text description before applying the regular expression by the RScript. The description text contains irrelevant character types like numbers , special symbols , spaces , etc.



After appying the five steps of filteration , now the text description is suitable or the next stage of transformation which is Document Term Matrix (DTM) . DTM is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In this case the DTM will tied to each of the TTS records.



## Create a Document Term Matrix (DTM)

Document Term Matrix (DTM) is a matrix based dataset which describes the frequency of terms that occur inside collection of dataset especially the text based types. The DTM was spesifically designed to have rows and columns. Rows is correspond as related unique rows records (*documents*) and columns as the unique terms which exist inside the selective column dataset.

The RScript below implements the Document Term Matrix function :-

mach\_corpus = Corpus(VectorSource(df\_review$review))  
 tdm = TermDocumentMatrix(mach\_corpus,control = list(removePunctuation = TRUE,  
 stopwords = c(stopwords("english")),  
 removeNumbers = TRUE, tolower = TRUE))  
 m <- as.matrix(tdm)

The description of R Script as follows :-

|  |  |
| --- | --- |
| R Script | Description |
| Corpus(VectorSource(df\_review$review)) | Representing and computing on corpora. |
| TermDocumentMatrix | Constructs or coerces to document-term matrix. |
| c(stopwords("english")) | Return various kinds of stopwords |
| removeNumbers = TRUE | Remove numbers from a text document |
| as.matrix(tdm) | Turn argument into a matrix |

Example of generated Document Term Matrix from TTS dataset as follows :-

Document Term Matrix of TTS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| tt\_num | access | alarm | already | atpmgitn |
| dfcc-1-12314522618 | 0 | 0 | 0 | 0 |
| dfcc-1-12314390988 | 0 | 0 | 0 | 0 |
| dfcc-1-12313258208 | 0 | 0 | 0 | 1 |
| dfcc-1-12312911038 | 0 | 0 | 0 | 0 |
| dfcc-1-12311810828 | 0 | 0 | 0 | 0 |
| dfcc-1-12311581758 | 0 | 0 | 0 | 0 |
| dfcc-1-12311375368 | 0 | 0 | 1 | 0 |
| dfcc-1-12310229828 | 0 | 1 | 1 | 0 |
| dfcc-1-12307977728 | 1 | 0 | 1 | 0 |
| dfcc-1-12306912058 | 0 | 0 | 0 | 0 |

## <<DocumentTermMatrix (documents: 20, terms: 103)>>  
## Non-/sparse entries: 393/1667  
## Sparsity : 81%  
## Maximal term length: 31  
## Weighting : term frequency (tf)

The information above summarize that the sparsity level is above 80% and removal of sparse term is needed for the next process . Some term like access , alarm , already , atpmgitn is low in frequency which proves all TTS desc sparsity level.

## Sparse Terms Removal

Sparse Terms is the term which has low frequency . This can help the generalization process and prevent overfitting into the model. The example show the R Script with removal of 60% of the sparsity level inside the TTS desc field.

tdm <- removeSparseTerms(tdm, sparse= 0.6 )

Sample of generated Document Term Matrix after removal of 60% (0.6) sparsity term below :-

DTM removeSparseTerms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| tt\_num | check | contact | downno | end |
| dfcc-1-12314522618 | 3 | 1 | 1 | 2 |
| dfcc-1-12314390988 | 1 | 0 | 1 | 1 |
| dfcc-1-12313258208 | 0 | 0 | 0 | 0 |
| dfcc-1-12312911038 | 3 | 1 | 0 | 2 |
| dfcc-1-12311810828 | 0 | 0 | 0 | 0 |
| dfcc-1-12311581758 | 2 | 1 | 0 | 2 |
| dfcc-1-12311375368 | 0 | 0 | 1 | 1 |
| dfcc-1-12310229828 | 0 | 0 | 1 | 1 |
| dfcc-1-12307977728 | 0 | 0 | 0 | 1 |
| dfcc-1-12306912058 | 2 | 1 | 0 | 2 |

## <<DocumentTermMatrix (documents: 20, terms: 17)>>  
## Non-/sparse entries: 220/120  
## Sparsity : 35%  
## Maximal term length: 14  
## Weighting : term frequency (tf)

After applying the remove sparse term filter , the DTM sparsity level was reduced to 35% . Now the DTM dimension is applicable as part of the predictive model.

## Vectorization of TTS (VTTS)

To enable dataset for prediction data processing, we need to convert all the independent variables inside the TTS which is in factor format (*code status*) into numeric format. In this sample we select only one of the fields which is the cause code as the sample for vectorizing the dataset. This technique will be applied to all non-numeric field formats inside the TTS dataset.

Sample TTS Cause Code

|  |  |
| --- | --- |
| tt\_num | cause\_code |
| dfcc-1-12315710868 | \* disconnect in internal wirin |
| dfcc-1-12315710637 | nt mat6d faulty |
| dfcc-1-12315707678 | \* sub system / cpe |
| dfcc-1-12315693158 | \* sub system / cpe |
| dfcc-1-12315693058 | others pcm |

To change the cause code into a vector , we need to apply the R Script code below to convert all the factor into numerical format.

for(i in names(df\_all)){  
 num <- as.numeric(as.factor(df\_all[,i]))  
 df\_all <- cbind(df\_all,num)  
 names(df\_all)[names(df\_all)=="num"] <- paste(names(df\_all[i]),"\_factor",sep = "")  
 }

as.numeric is a function which creates the objects into numeric and as.factor encodes a vector as a factor. A table below shows the result of the encoding from the cause code factor status into the numerical format.

Sample TTS Cause Code Vectorization

|  |  |  |
| --- | --- | --- |
| tt\_num | cause\_code | cause\_code\_factor |
| dfcc-1-12315710868 | \* disconnect in internal wirin | 1 |
| dfcc-1-12315710637 | nt mat6d faulty | 110 |
| dfcc-1-12315707678 | \* sub system / cpe | 9 |
| dfcc-1-12315693158 | \* sub system / cpe | 9 |
| dfcc-1-12315693058 | others pcm | 126 |

# Results

## VTTS with DTM

The final steps of this techniques is to combine all the vectorization of TTS with the DTM. All the standard name of each selected column will be rename into [fieldname]\_factor format. Example shows below :-

|  |  |
| --- | --- |
| Original column name | Vector column name |
| created\_by | created\_by\_*factor* |
| account\_num | account\_num\_*factor* |
| product\_factor | product\_*factor* |
| sub\_product | sub\_product\_*factor* |
| symp\_err | symp\_err\_*factor* |
| ... | ... |

The construction design of the table structure as follows :-

* Select the tt\_num (*ticket number*) columns. Tt\_num is the key of the TTS dataset.

df\_tt\_num <- as.data.frame(queryResult$tt\_num)

* Selecting all the factor columns

df <- as.data.frame(queryResult)  
 names(df)[1] <- "created\_by\_factor"  
 names(df)[2] <- "account\_num\_factor"  
 names(df)[..n] <- "[..x]\_factor"  
 ....  
 replace [..n] with column number and [..x] is the column name  
 ....

* Binding all the factor columns with the tt\_num

df <- cbind(df\_tt\_num,df)

* The results after binding tt\_num with some of the factor columns and the tdm matrix

Ticket No , TTS Vectorization and TDM table (continued below)

|  |  |  |
| --- | --- | --- |
| tt\_num | nova\_account\_num\_factor | account\_num\_factor |
| 1-10649308447 | 1 | 20 |
| 1-10649125427 | 12 | 19 |
| 1-10649464881 | 92 | 18 |
| 1-10649636010 | 58 | 31 |
| 1-10649267567 | 83 | 1 |

Table continues below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [x]\_factor | account | acct | cable | check | created |
| ... | 0 | 1 | 1 | 1 | 1 |
| ... | 0 | 0 | 4 | 0 | 0 |
| ... | 1 | 1 | 1 | 1 | 2 |
| ... | 2 | 1 | 1 | 1 | 1 |
| ... | 3 | 1 | 0 | 0 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| cust | customer | name | next | numberlogin | service |
| 1 | 2 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 0 | 0 | 1 |
| 1 | 2 | 1 | 1 | 1 | 1 |
| 0 | 5 | 0 | 2 | 1 | 1 |
| 0 | 2 | 1 | 0 | 1 | 2 |

The table above show that combines three elements which is TT\_NUM , TTS vector and the DTM matrix. Now all the field is ready to utilize for further analysis such as classification and prediction .

# References

N.p., n.d. Web.  
Ye, Yangdong, Jing Zhang, Junwei Gao, and Limin Jia.   
"The Application of Decision Tree Induction of Classification in Train   
Tickets System." Proceedings. International Conference on Machine Learning and   
Cybernetics (2002): n. pag. Web.  
  
Ye, Yangdong, Jing Zhang, Junwei Gao, and Limin Jia. "The Application of Decision  
Tree Induction of Classification in Train Tickets System."Proceedings. International   
Conference on Machine Learning and Cybernetics (2002): n. pag. Web.  
  
Marcu, Patricia, Genady Grabarnik, Laura Luan, Daniela Rosu, Larisa Shwartz, and Chris Ward.  
"Towards an Optimized Model of Incident Ticket Correlation."2009 IFIP/IEEE   
International Symposium on Integrated Network Management(2009): n. pag. Web.  
  
Shao, Qihong, Yi Chen, Shu Tao, Xifeng Yan, and Nikos Anerousis. "EasyTicket." Proc.   
VLDB Endow. Proceedings of the VLDB Endowment 1.2 (2008): 1436-439. Web.