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 technique of data processing for the ease for the next stage of classification and prediction.On the other hand , we applied the new techniques of data mining by combining with the Bigdata elements for faster and achievable results.We realize that the combination of this techniques will provide luxuriant decision information for any organization and management.It also can become a fulfillment the field of data mining

1 Keywords

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

2 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.

3 Related work

3.1 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.

3.2 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 :-

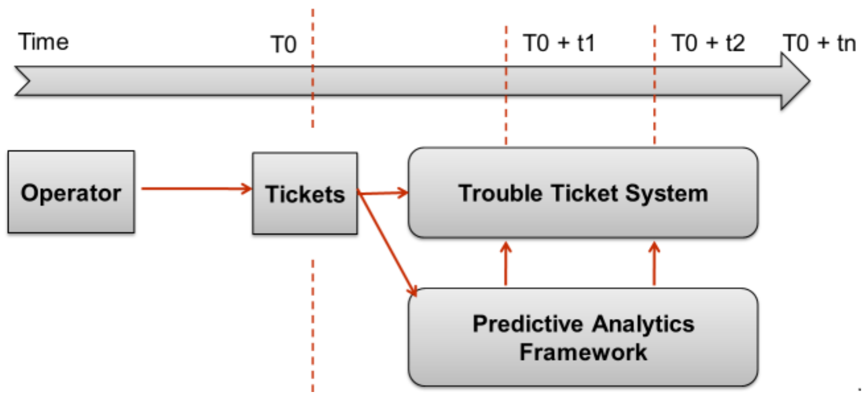


Figure 1: Trouble ticket system with predictive analytics framework

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.

4 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)
- VTTS and DTM
- Training
- Testing
- Prediction

4.1 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
```

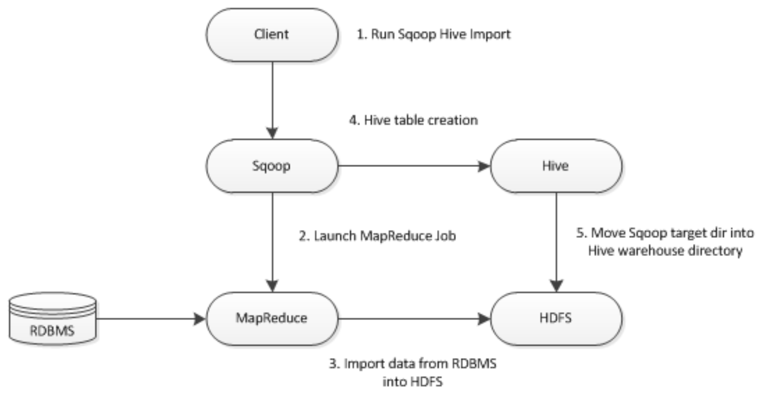


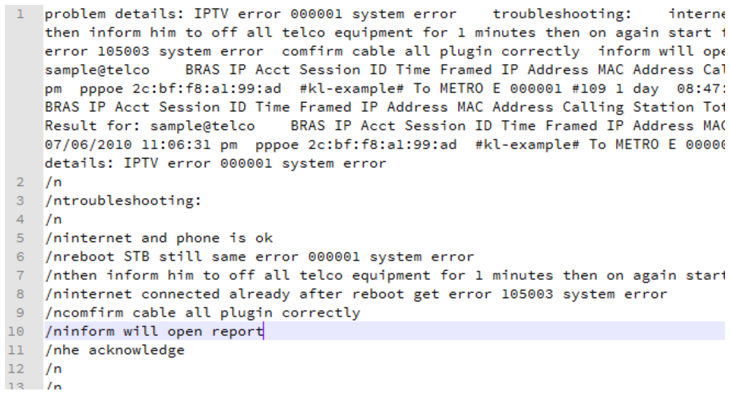
Figure 2: Hadoop - Sqoop - Hive Operation

```
-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 :-

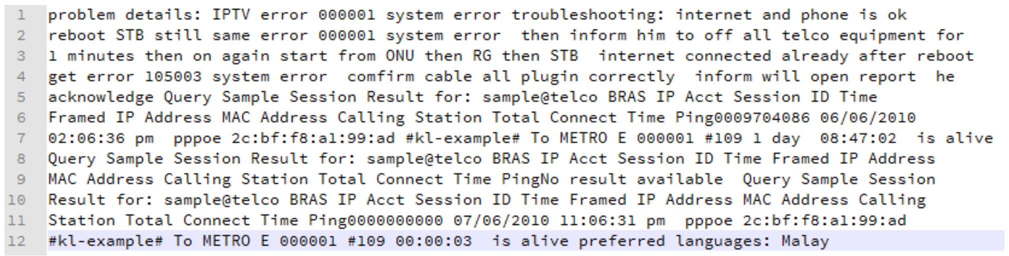
Syntax	Description
sqoop import	Import a table from a database to HDFS
–connect jdbc:[rdbms driver]	Specify JDBC connect string
[ip_address]:[port]/[db_name]	Provides ip address , port no and rdbms database name
–username [username]	Provides the username
–password [password]	Provides the password
–target-dir [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
‘[ˆa-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 :-



```
problem details: IPTV error 000001 system error    troubleshooting:    interne
then inform him to off all telco equipment for 1 minutes then on again start f
error 105003 system error  comfirm cable all plugin correctly  inform will ope
sample@telco    BRAS IP Acct Session ID Time Framed IP Address MAC Address Cal
pm  pppoe 2c:bf:f8:a1:99:ad  #kl-example# To METRO E 000001 #109 1 day  08:47:
BRAS IP Acct Session ID Time Framed IP Address MAC Address Calling Station Tot
Result for: sample@telco    BRAS IP Acct Session ID Time Framed IP Address MAC
07/06/2010 11:06:31 pm  pppoe 2c:bf:f8:a1:99:ad  #kl-example# To METRO E 00000
details: IPTV error 000001 system error
/n
/ntroubleshooting:
/n
/ninternet and phone is ok
/nreboot STB still same error 000001 system error
/nthen inform him to off all telco equipment for 1 minutes then on again start
/ninternet connected already after reboot get error 105003 system error
/ncomfirm cable all plugin correctly
/ninform will open report
/nhe acknowledge
/n
/n
```

Figure 3: Original - (desc) field text



```
problem details: IPTV error 000001 system error troubleshooting: internet and phone is ok
reboot STB still same error 000001 system error  then inform him to off all telco equipment for
1 minutes then on again start from ONU then RG then STB  internet connected already after reboot
get error 105003 system error  comfirm cable all plugin correctly  inform will open report  he
acknowledge Query Sample Session Result for: sample@telco BRAS IP Acct Session ID Time
Framed IP Address MAC Address Calling Station Total Connect Time Ping0009704086 06/06/2010
02:06:36 pm  pppoe 2c:bf:f8:a1:99:ad #kl-example# To METRO E 000001 #109 1 day  08:47:02  is alive
Query Sample Session Result for: sample@telco BRAS IP Acct Session ID Time Framed IP Address
MAC Address Calling Station Total Connect Time PingNo result available  Query Sample Session
Result for: sample@telco BRAS IP Acct Session ID Time Framed IP Address MAC Address Calling
Station Total Connect Time Ping0000000000 07/06/2010 11:06:31 pm  pppoe 2c:bf:f8:a1:99:ad
#kl-example# To METRO E 000001 #109 00:00:03  is alive preferred languages: Malay
```

Figure 4: Filtered - (desc) structure after applying REGEXP_REPLACE during `Sqoop` operation

4.2 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 :-

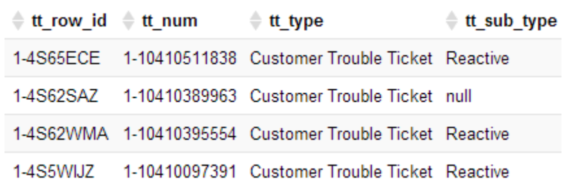
```
drop function IF EXISTS row_sequence();
create function IF NOT EXISTS row_sequence()
returns BIGINT location 'lib/hive-contrib-0.12.0-cdh5.1.2.jar'
symbol='org.apache.hadoop.hive.contrib.udf.UDFRowSequence';
```

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;
```

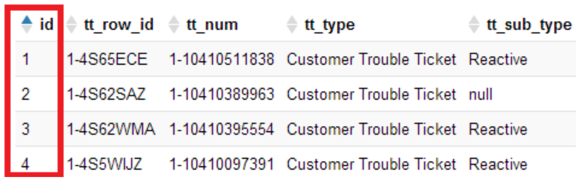
```
create table IF NOT EXISTS tts_stage1 as select row_sequence() as id ,
* from [tts_table] limit 10000;
```

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`



tt_row_id	tt_num	tt_type	tt_sub_type
1-4S65ECE	1-10410511838	Customer Trouble Ticket	Reactive
1-4S62SAZ	1-10410389963	Customer Trouble Ticket	null
1-4S62WMA	1-10410395554	Customer Trouble Ticket	Reactive
1-4S5WIJZ	1-10410097391	Customer Trouble Ticket	Reactive

Figure 5: Initial Hive TTS Dataset after Sqoop operation

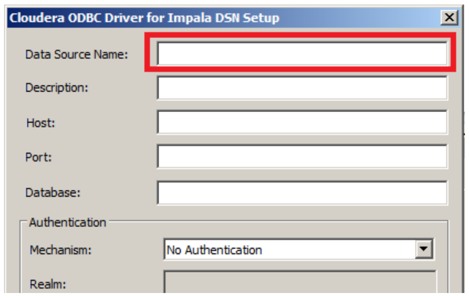


id	tt_row_id	tt_num	tt_type	tt_sub_type
1	1-4S65ECE	1-10410511838	Customer Trouble Ticket	Reactive
2	1-4S62SAZ	1-10410389963	Customer Trouble Ticket	null
3	1-4S62WMA	1-10410395554	Customer Trouble Ticket	Reactive
4	1-4S5WIJZ	1-10410097391	Customer Trouble Ticket	Reactive

Figure 6: Posterior Hive TTS Dataset after applying Hive-SQL

4.3 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).



Cloudera ODBC Driver for Impala DSN Setup

Data Source Name:
Description:
Host:
Port:
Database:
Authentication
Mechanism: No Authentication
Realm:

Figure 7: Hive ODBC Datasource Administrator

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 :-

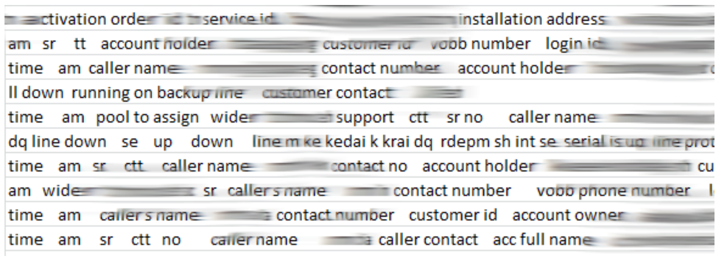
```
library(RODBC)
conn <- odbcConnect(dsn="[dsn_name]", uid="[username]", pwd="[username]") #
queryResult <- sqlQuery(conn, "select desc from [db_name].[hive_table_with_id]
where id > 0 and id <= 10000 ")
odbcClose(conn)
```

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$review))`
2	Remove all the numbers	`df_desc <-as.data.frame (sapply(df_desc,gsub,pattern="[[:digit:]]",replacement=""))`
3	Remove all the punctuation and symbols	`df_desc<-as.data.frame (sapply(df_desc,gsub,pattern="[[:punct:]]",replacement=""))`
4	Trim the unnecessary spaces	`df_desc <-as.data.frame (sapply(df_desc,gsub,pattern="@\\w+",replacement=""))`
5	Trimming the whitespace	`df_desc <-as.data.frame (sapply(df_desc,gsub,pattern="^\\s+|\\s+$",replacement=""))`

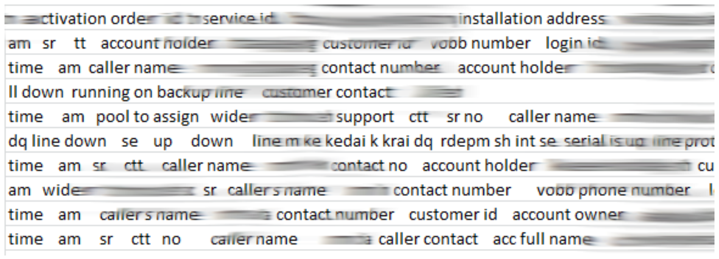
The image below shows state of text description before applying the regular expression by the RScript. The description text contains irrelevant character types like numbers , special symbols , spaces , etc.



```
[redacted]ctivation order [redacted] service id [redacted] installation address [redacted]
am  sr  tt  account holder [redacted] customer id  vobb number  login id [redacted]
time  am caller name [redacted] contact number  account holder [redacted]
ll down running on backup line  customer contact [redacted]
time  am  pool to assign  wide [redacted] support  ctt  sr no  caller name [redacted]
dq line down  se  up  down  line m ke kedai k krai dq rdepm sh int se serial is up line prot
time  am  sr  ctt  caller name [redacted] contact no  account holder [redacted] cu
am  wide [redacted] sr  caller s name [redacted] contact number  vobb phone number  l
time  am  caller s name [redacted] contact number  customer id  account owner [redacted]
time  am  sr  ctt  no  caller name [redacted] caller contact  acc full name [redacted]
```

Figure 8: description text on before filteration

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.

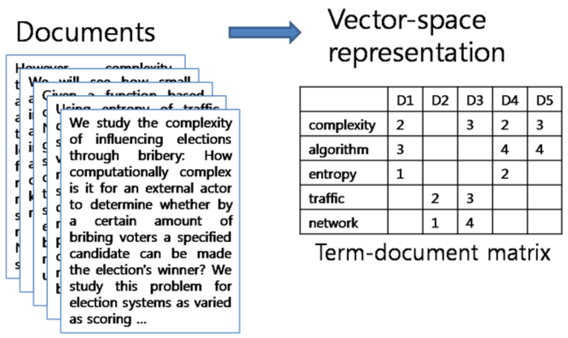


```
[redacted]ctivation order [redacted] service id [redacted] installation address [redacted]
am  sr  tt  account holder [redacted] customer id  vobb number  login id [redacted]
time  am caller name [redacted] contact number  account holder [redacted]
ll down running on backup line  customer contact [redacted]
time  am  pool to assign  wide [redacted] support  ctt  sr no  caller name [redacted]
dq line down  se  up  down  line m ke kedai k krai dq rdepm sh int se serial is up line prot
time  am  sr  ctt  caller name [redacted] contact no  account holder [redacted] cu
am  wide [redacted] sr  caller s name [redacted] contact number  vobb phone number  l
time  am  caller s name [redacted] contact number  customer id  account owner [redacted]
time  am  sr  ctt  no  caller name [redacted] caller contact  acc full name [redacted]
```

Figure 9: description text on after filteration

4.4 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.



We study the complexity of influencing elections through bribery: How computationally complex is it for an external actor to determine whether by a certain amount of bribing voters a specified candidate can be made the election's winner? We study this problem for election systems as varied as scoring ...

	D1	D2	D3	D4	D5
complexity	2		3	2	3
algorithm	3			4	4
entropy	1			2	
traffic		2	3		
network		1	4		

Figure 10: Document Term Matrix

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 :-

Table 8: 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.

4.5 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 :-

Table 9: 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.

4.6 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 the course code as the sample for vectorize the dataset. This technique can be applied to all non-numeric formats of the other field inside the TTS dataset.

Table 10: 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 applied the

```
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 = "")
}
```

Table 11: Sample TTS Cause Code and 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

4.7 VTTS with DTM

Nothing here

5 Experiments and results

Nothing here

5.1 Training

Nothing here

5.2 Testing

Nothing here

5.3 Prediction

Nothing here

5.4 Results

Nothing here

6 Acknowledgments

Nothing here

7 References

Nothing here