PROJECT REPORT:

EMAIL RATING

OCTANE MARKETING PVT. LTD.

HARI ESWAR SM
IIT ROORKEE

ELECTRICAL ENGINEERING, 3rd YEAR

ENGINEERING INTERN

Introduction

The client provides us with a list of email IDs for their campaign. This number is small when compared to our already existing database of email IDs. For example, we already have a database of bounce email IDs (email IDs where if you send a mail, it bounces back) and we need to **cross check** if the list of email IDs the client gave us has some IDs in common with the database of bouncy emails which we already have. Optimizing this process is the goal.

Conventional search methods wouldn't work because the database which we have vs list which we are given is a very large number (Crores vs Lakhs). We need a way to optimize this problem.

Before the above process, we need to classify the database of available email IDs into five categories:

1. Super active
2. Active
3. Normal
4. Inactive
5. Bounce

The raw data which we have is available with the following details:

1. First Open info – First open timestamp for the mail ID for a particular campaign
2. Click info – A timestamp that records every time the user clicks the campaign which we send him
3. Campaign Sent Time – What time is a particular campaign sent.
4. Device – Can be multiple devices held by the same consumer
5. IP – to help us in geolocation:
   i. Opening IP address
   ii. Click IP address (both of these can be different)
6. Client ID – unique
7. Campaign ID – unique for every campaign
8. Platform – which web service do we choose to send the mail (?)
9. User ID – what we login with, a user can have many clients
10. Click URLs – what are the URLs that are clicked by the consumer
11. Subject of the mail

Classifying the active email IDs

In the first part of the problem, we work out the active email ID. It is found that about 15-20 % of the emails in the upper limit are only responsive to the emails which we send them. And out of the entries only 2-3% only register as a click. Classification of these emails is done as:

1. Normal
2. Active
3. Superactive

Structure of Data available

The data is available in many tables in the following formats:

config_table				
platform	**userid**	**date_from**	**date_to**	**db_info**

In the above table, the information of the database where the data is stored is extracted. db_info stores the details of access username, password and host address. We can get a specific info by specifying a range of dates (for example if we need db_info between Jan and March, etc.).

Platform is the platform through which the user sends their campaign.

From db_info we get the following tables in the database:

email_bank	
email_key	**emailid**

An email_key is a unique key that will help in hashing the details corresponding to the mail ID directly. This will have time optimization.

campaign_table		
platform_campaign_id	**campaign_sent_time**	**subject**

Platform Campaign ID will be unique for every campaign a user/client sends. A user can have many clients. campaign_sent_time will be constant for a particular campaign. There is a processing error in this because our campaigns go through a Message Transfer Agent.

campaign_sent_table
emaild (NOT NULL)

Campaign Sent Table consists of the emailids the particular campaign was sent to. For a particular campaign, the

open_activity_table					
emailid	**platform_campaign_id**	**first_open_time**	**total_opens**	**device**	**ip**

Email Open Activity is recorded here. first_open_time will be the first time the email ID opens the mail of the particular campaign. last_open_time will be the most recent open with respect to the campaign. It keeps updating when the user **clicks** a mail.

The record of the number of times opened is also available.

click_activity_table

platform_campaign_id	clicked_time	click_url	device	ip

click_url is the URL the consumer clicks in the campaign. There can be multiple URLs for the same campaign and the user can click many such URLs.

Required data (after processing)

a. For classifying email IDs based on activity

The data required for sorting the email IDs based on activity are:

1. Open Rate
2. Unique Open Rate
3. Click-through Rate
4. First Open Time – Campaign Sent Time

b. For finding the best time to send an email

The data required for sorting the email IDs' best opening time are:

1. List of all open times
2. List of all click times

Approach to Part (a) of the Problem

The problem consists of two parts:

1. Reducing Redundancies in the existing data.
2. Classification of the email IDs based on their activity.

Documentation

1. Reducing Redundancies in existing Data

The following features are considered for calculating the email activity:

1. Open Rate
2. Click Rate
3. Unique Open Rate
4. First Open Time – Campaign Sent Time

1. **Open Rate** – Ratio of the number of times the email ID has opened all the campaigns it has received to the number of mails sent to the email ID.
2. **Click Rate** – Ratio of the number of times the email ID has clicked a link that was sent in the email for a campaign to the number of mails sent to the email ID.
3. **Unique Open Rate** – Ratio of the unique campaigns the email ID has opened to the number of mails sent to the email ID.
4. **First Open Time – Campaign Sent Time** – The minimum value for First Open Time – Campaign Sent Time for all the campaigns that were sent to the email.

Initially the following two features were also considered:

1. Last Open Time – First Open Time
2. Click Time – Campaign Sent Time

However the above two features are not an exact measure of the activity of the email ID. It is because campaigns visited when an email ID's inbox is being cleaned up will also count as Last Open Time and the percentage of such emails were found to be high and abnormal values skewed the classification results. Click Time – Campaign Sent Time also skews the data because there are a lot of cases where the user has clicked on a campaign URL after a year of sending the campaign. Also if the email ID has even clicked the campaign, it is considered to be a measure of activeness, hence it is not advisable to take it as a feature because if the user clicks the URL after a long time (say more than a month) such features may get clustered together.

The following programs perform the entire process of reducing the redundancies, all of the programs have been written in PHP:

1. Load Data (load_data.php) – This program consists of the functions to delete the existing data and load new data into the three tables – campaign_sent_table, open_activity_table and click_activity_table and to sort the data based on email ID.
2. Open Collect (open_collect.php) – This program consists of the function to remove the redundancies within the open_activity_table's email IDs. The final product will be the email ID, summation of the email ID's individual total opens per campaign and the number of campaigns the email ID has opened (i.e. total number of entries of the email ID in the open_activity_table) and the least first open minus campaign sent time.

3. Click Collect (click_collect.php) - This program consists of the function to remove the redundancies within the click_activity_table's email IDs. The final product will be the email ID and the number of campaigns the email ID has clicked (i.e. total number of entries of the email ID in the click_activity_table).
4. Sent Collect (sent_collect.php) – This program consists of the function to remove the redundancies within the campaign_sent_table's email IDs. The final product will be the email ID and the number of campaigns the email ID has received (i.e. total number of entries of the email ID in the campaign_sent_table).
5. Reduce Program (ultimate_reduce.php) – Ultimately the program has the function that will create a .csv file that will contain the features that will be taken for the classification.
6. Configuration (config.php) – This consists of the various variables in the program that will help in the procedural flow of the program. Change the variables here to change the various variables like Input Folder, Output Folder, and Decisions in the various points of the program. This consists of the various variables in the program that will help in the storage and database details like table names, username, password and hostname for the database.
7. Test (test.php) – Coordinates all of the program together. It takes two arguments – the name of the config.php file and the user id from the bash function.

2. Classification of the email IDs based on their activity

K-Means clustering is employed here. It is a method of Unsupervised Machine Learning. The program is run on python and requires the following modules:

i. Numpy + MKL
ii. Scipy
iii. SKLearn (SciKit Learning)

The following program is written in Python for classification:

> classify.py – A python script that will perform the classification process that takes the user id as an input argument.

The programs are all coordinated and made to run with a bash file run.sh (this bash file will also run the classification algorithm once the redundancies are removed).

Initial Approach – Reduction of Data

The problem can be divided into the following steps:

1. Deciding on the required architecture
2. Deciding on the constraint that will not cost us the most
3. Deciding the reduction algorithm to get our features
4. Classification

Initially, the approach was viewed from MySQL perspective.

The features that were decided initially were with respect to Open Activity Data and Click Activity Data only. The following features were thought to be appropriate measures for deciding the email activity:

1. Number of Opens (Sum of the total opens for a campaign by an email ID across all the campaigns sent to it)
2. Number of Clicks (Number of occurrences of the email ID in the click activity table)
3. First Open Time – Campaign Sent Time (The minimum of which is taken for the email ID as a measure of how quickly the email ID opens the campaign sent to it)
4. Last Open Time – First Open Time (The least of which is taken for all campaigns the email ID has received, however this was immediately scrapped because of the reason mentioned before - It is because campaigns visited when an email ID's inbox is being cleaned up will also count as Last Open Time and the percentage of such emails were found to be high and abnormal values skewed the classification results)
5. Click Time – Campaign Sent Time (The least of which is taken for all campaigns the email ID has received, later this was also dropped because of the reason mentioned before - there are a lot of cases where the user has clicked on a campaign URL after a year of sending the campaign. Also if the email ID has even clicked the campaign, it is considered to be a measure of activeness, hence it is not advisable to take it as a feature because if the user clicks the URL after a long time (say more than a month) such features may get clustered together)

This meant that if an email ID was to open a campaign more than 5 times (say), the email ID would likely be classified as Active or Superactive. Taking this approach in mind, we used the following MySQL queries through PHP PDO connection to extract the data like so,

```
SELECT emailid, TIMESTAMPDIFF(second, campaign_sent_time, first_open_time) AS
first_minus_sent, total_opens FROM open_activity_table INNER JOIN
campaign_table ON campaign_table.campaignid = open_activity_table.campaignid;
```

For extraction of data required and store the row wise result as an array `$reduce` that will be dynamically allocated using PHP associative arrays where the row field will be referenced by the email address and the column will consist the required features. The entire program can be found in reduce_program.php file.

Note: PHP also has a maximum memory limit it can process at a time in its program. This was found to be approximately equal to 15 00 000 (15 lakh) entries in the reduce array. Since we have data of more than this number, batch processing was done after ordering the email ID using `LIMIT` statement in MySQL query. This can also be found in reduce_program.php file. We set a `$startlimit` and a `$batchsize` and then use `ORDER BY emailid LIMIT $startlimit, $batchsize;` at the end of our above query.

It should also be noted to store the last row of the present batch to make sure it doesn't occur in the next batch.

Initial Approach – Inactivity Data

A part of the project is to get the list of email IDs that do not have an entry in the open activity table. A direct SQL query will be to do a sub query in the WHERE clause as so:

```
SELECT emailid FROM campaign_sent_table WHERE emailid NOT IN (SELECT emailid
FROM open_activity_table);
```

However the statement will take a very long time to execute because the subquery will take a long time to compile together and then every entry of the campaign_sent_table would have to be cross referenced with the entire size of the open_activity_table. For eg. if the open_activity_table has 5-7% of the entries in the campaign_sent_table and there are 50000 entries in open_activity_table then the scale of the comparison is 1 000 000 vs 50 000 which will be a long time to compute and give the results. Also real data will be even bigger than 1 000 000 for the sent table. Hence this method isn't advisable to use on such a large data set.

Hence batch processing was used.

The difference would be to query batches of email IDs from open_activity_table and concatenate them as a string which would replace the sub query. Direct querying of open_activity_table entries would be taking less time and thus if indexing is present then this process would be even faster.

The code snippet is as follows:

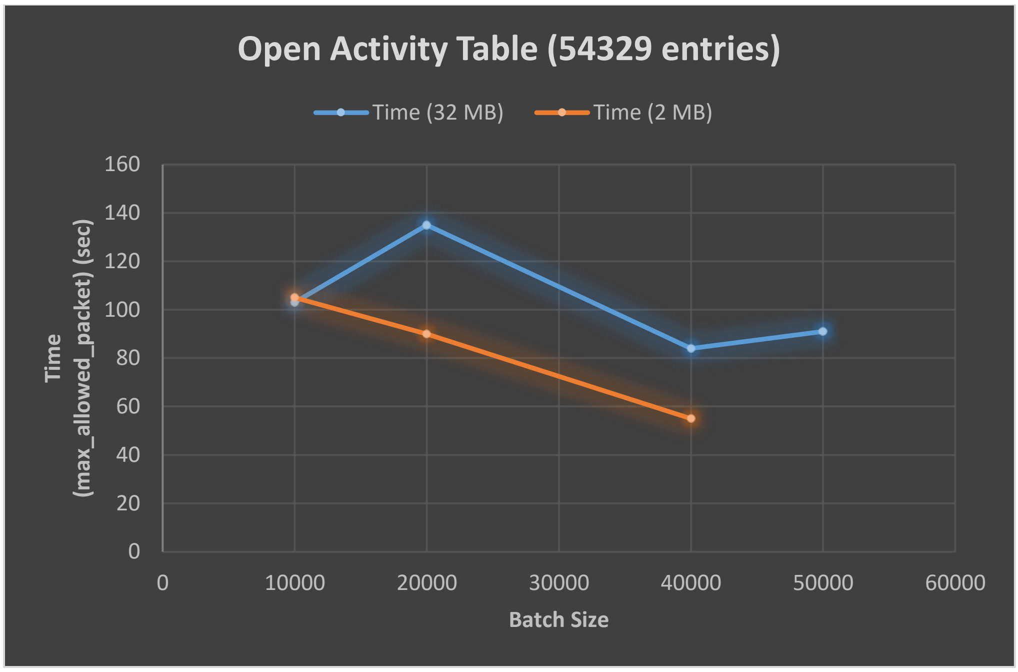
```
public function push_in_batch($emailid){

                  $this->counter++;

                  $this->main_counter++;

                  if($this->counter==$this->max_batch_limit){

                        $this->email_str = $this->email_str."'".$emailid."'";

                  }

                  else{

                        $this->email_str = $this->email_str."'".$emailid."',";

                  }

            }
```

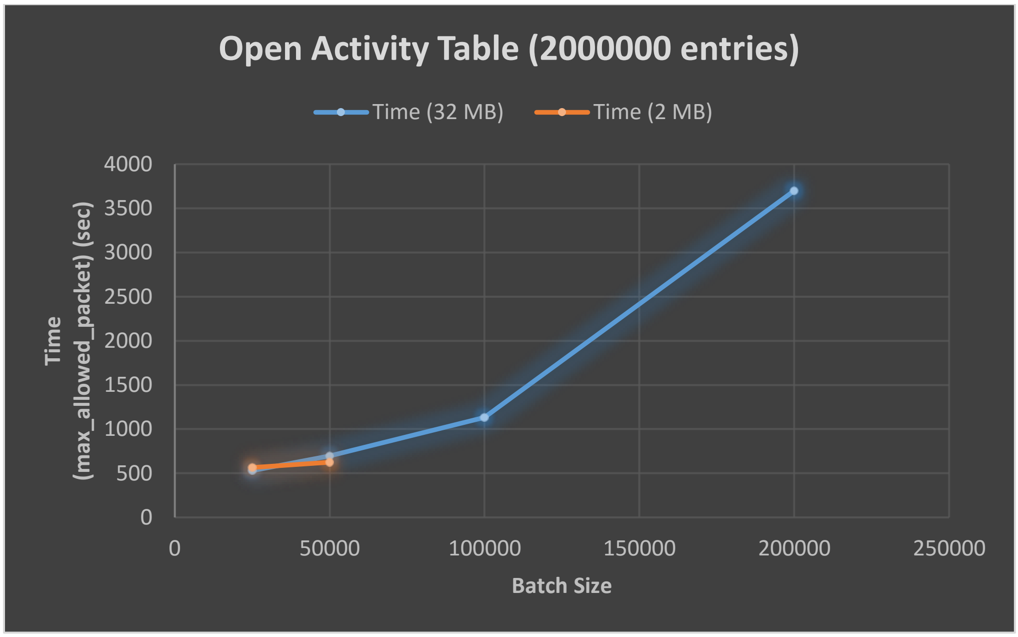
We use a main counter and another counter. The main counter counts the number of batches passed and will stop when all of the data in the open activity table is processed while the counter checks the batch currently being processed and goes till the batch size (which is fixed is reached).

Some variables that will have to be changed in MySQL configuration itself. They are:

1. max_allowed_packet size – The size of the string that can be held in the WHERE clause. Evidently the size of the batch is computed and if it is larger than this max_allowed_packet size then it is found that MySQL error of "MySQL has gone away" occurs.
2. innodb_buffer_pool_instances – Should be set to be less for more RAM to be allocated for the process.

After using the default innodb_buffer_pool_instances of 8, the following data is obtained with using different batch sizes for different max_allowed_packet sizes.





Figures: The graphical representation of the time taken for different batch sizes and the different max_allowed_packet sizes used.

```
$sql = "SELECT emailid FROM clone_activity_table";

$res = $dbh->query($sql);

foreach($res as $row){

                   $emailid = $row['emailid'];

                   $obj->push_in_batch($emailid);
```

```
                    if($obj->is_batch_complete()){
                                    $arr = $obj->get_completed_batch();
                                    $obj->email_str  =  null;
                                    $sth = $dbh->query("DELETE FROM clone_sent_table 
WHERE emailid IN ($arr);");
                    }
            }
```

The above is the main program for deleting the common entries. Notice that after every batch email_str is emptied. This is important or else the batch size will keep increasing in size and will exceed the max_allowed_packet size.

Once the loop has ended, we need to call the get_completed_batch() function and then the delete function to get the last batch processed that has a batch size less than the batch size specified as a constant.

Initial Approach – Counting the email ID occurrences

Ordering of the data and producing count for the corresponding email IDs was found to consume time.

A count(emailid) query in MySQL would remove the need for the separate collect programs for sent, open and click tables respectively. However a count(*) query does three things:

1. Orders the email ID
2. Collects the count and assigns with the particular email ID
3. Prints the query (incase a SELECT statement is attached to it)

Hence it is not advisable to use the count query to get the various values like Unique Opens or Total Number of campaigns sent, etc.

The reduce program was written initially without considering the sent details for an email ID, however this was found to be an inaccurate way of measuring email ID activity. Sent information plays as much as a role in deciding the activity of an email ID. An email ID that has received 100 campaigns of which it has opened 10 is **less active** than an email ID that has opened all 6 of the campaigns it has received.

But processing all of the data by MySQL is not time efficient. Using methods like INNER JOIN and LEFT JOIN or some combination of both will not prove to be time efficient. Hence we adapt the method of file processing for getting the count details.

File Processing approach

The data is initially truncated as per the user's choice from the tables. Ordering and extraction is done on the MySQL side with the help of the following queries:

1. For open_activity_table's features – Required are email ID, total opens, first open time minus campaign sent time. Hence we INNER JOIN open activity data with the campaign_table data that

consists of the campaign sent time matching on campaign id. TIMESTAMPDIFF function returns the difference in times in seconds.

```
SELECT emailid, TIMESTAMPDIFF(second, campaign_sent_time, first_open_time) AS
first_minus_sent, total_opens FROM open_activity_table INNER JOIN
campaign_table ON campaign_table.campaignid = open_activity_table.campaignid
ORDER BY emailid into outfile 'E:/open_activity_data.csv' fields enclosed by
'"' terminated by ',' lines terminated by '\n';
```

2. For click_activity_table's features – Only email ID is required. Hence we inner join click activity data with the campaign_table data that consists of the campaign sent time. TIMESTAMPDIFF function returns the difference in times in seconds, we are taking click time minus campaign sent time into the following MySQL query in case we need it for any other purpose.\

```
SELECT emailid, TIMESTAMPDIFF(second, campaign_sent_time, click_time) AS
click_minus_sent FROM click_activity_table INNER JOIN campaign_table ON
campaign_table.campaignid = click_activity_table.campaignid ORDER BY emailid
into outfile 'E:/click_activity_data.csv' fields enclosed by '"' terminated
by ',' lines terminated by '\n';
```

3. For campaign_sent_table's features – Only email ID is required. Hence a simple order by command is executed to be written into the CSV file.

```
SELECT emailid FROM campaign_sent_table ORDER BY emailid into outfile
'E:/sent_data.csv' fields enclosed by fields enclosed by '"' terminated by
',' lines terminated by '\n';
```

File processing is fast and doesn't require any additional software. So all of the calculations were better to be done on the file end.

Ordering the sent data, open data and click data on the MySQL side and then count of email ID and the other vital information was to be calculated into the reduced data at the file side itself.

load_data.php

load_data.php is a class file **Load** used to initialize the database basically for the file processing later on. Based on the variables in the config.php file. It consists of the following functions:

1. delete() – deletion of the data in manual mode. Here, TRUNCATE TABLE is used. Truncate table is the fastest way to delete the contents of the table in MySQL. Other methods like DELETE FROM is slow because it needs to compile the data and will perform delete one by one. Truncate however just empties the contents of the table.
2. delete_auto() - deletion of the data in auto mode based on the variable's statuses in the config.php file.
3. load_sent_data() – Loading from a specific location in manual mode. Since this function runs in the manual mode, it involves the user typing the location of the directory that contains the input data. There is a flag input that is taken after a prompt presented to the user. A user can choose

to proceed with a 'Y' choice as input to the function. The input directory should contain the following branches:

   a. DIR/input/sent – the list of csv files that contain the campaign sent data (each file consists of only email IDs)
   b. DIR/input/open – the list of csv files that contain the campaign open data (each file consists of the standard format list of attributes as given in the database architecture above).
   c. DIR/input/click – the list of csv files that contain the campaign click data (each file consists of the standard format list of attributes as given in the database architecture above).

4. load_sent_data_auto() – Loading from the input location as given in the CSV file. There is a separate variable for the sent data input location. A glob of files is created in the directory that have the extension *.csv and all are read one by one by the program.
5. load_data() – Loading the data for open_activity_table and click_activity_table from the directory in manual mode. Since it is manual mode, the directory is taken as an input after a prompt is presented to the user. A user can choose to input only open or only click or both inside this program as they are taken as separate inputs. There is a flag outside of the program that will decide if the user wants to proceed with this function after getting an input from the user. A user can choose to proceed with a 'Y' choice as input to the function.
6. load_data_auto_open() – Loading the data for open_activity_table automatically based on the flag taken from a constant in the config.php file. A user can choose not to go ahead with loading open_activity_table data from the variable present in the config.php file.
7. load_data_auto_click() – Loading the data for click_activity_table automatically based on the flag taken from a constant in the config.php file. A user can choose not to go ahead with loading click_activity_table data from the variable present in the config.php file.
8. put_sort_csv_sent() – Sorting the campaign_sent_table and putting into a csv file for auto mode. (The manual mode functions for sorting and writing into a csv file are present in the class files of the respective tables.) The function used uses a MySQL query as given above.
9. put_sort_csv_open() – Sorting the open_activity_table and putting into a csv file for auto mode. (The manual mode functions for sorting and writing into a csv file are present in the class files of the respective tables.) The function used uses a MySQL query as given above.
10. put_sort_csv_click() – Sorting the click_activity_table and putting into a csv file for auto mode. (The manual mode functions for sorting and writing into a csv file are present in the class files of the respective tables.) The function used uses a MySQL query as given above.
    There is a sub function inside this function to delete the data which is present in click_activity_table that is not present in the open_activity_table. It is controlled by the DELETE_UNCOMMON variable in the config.php file. Such a case of an entry being in click_activity_table not there in open_activity_table might arise because of any problem when querying and extracting data from the slave itself.

sent_collect.php

sent_collect.php is a class file **Load_Sent** that consists of three functions – put_sort_csv(), reduce_to_count() and reduce_to_count_auto(). put_sort_csv() is used to sort the data and put it into the csv file for the reduce function in manual mode. reduce_to_count() is the reduction function for manual mode and reduce_to_count_auto() is the reduction function for auto mode.

For the reduction function

The algorithm goes like so:

```
while(!feof($sent_data_file)){
                                $row = fgetcsv($sent_data_file);
                                if(feof($sent_data_file))
                                        break;;
                                if($overall_count == 0){
                                        $prev = $row[0];
                                        $overall_count = $overall_count+1;
                                }
                                $now = $row[0];
                                if(strcmp($prev, $now) != 0){
                                        $sent_reduced_array[0] = $prev;
                                        if($overall_count==1){
                                                $sent_reduced_array[1] = $count;
                                        }
                                        else{
                                                $sent_reduced_array[1] = $count+1;
                                        }
                                        fputcsv($count_file, $sent_reduced_array);
                                        $prev = $now;
                                        $count = 0;
                                        $overall_count = $overall_count+1;
                                }
                                else{
                                        $count = $count + 1;
                                }
                        }
```

Maintain a count that will increment if the present email ID pointer matches with the previous pointer. If the email ID of the present pointer does not match with the previous pointer, then reset the counter to 0 and write the previous data to a csv file that will be used for file processing.

fgetcsv and fputcsv functions are used to read and write from the csv files respectively.

After the main loop is completed it is required to write once again into the csv file in order to write the last record. The exit condition is reached if end of file is reached. Since the last record of the file is the end of the file, it is necessary to write to the file one more time after the loop has ended.

open_collect.php

open_collect.php is a class file **Load_Open** that consists of three functions – put_sort_csv(), reduce_to_count() and reduce_to_count_auto(). put_sort_csv() is used to sort the data and put it into the csv file for the reduce function in manual mode. reduce_to_count() is the reduction function for manual mode and reduce_to_count_auto() is the reduction function for auto mode.

For open activity data, total opens and getting the least first open minus campaign sent time is also necessary. Additional pointers are used to maintain them separately.

The following is the code snippet for this:

```
while(!feof($open_data_file)){
                                $row = fgetcsv($open_data_file);
                                if(feof($open_data_file))
                                        break;
                                if($overall_count == 0){
                                        $prev = $row[0];
                                        $first_minus_sent_prev = $row[1];
                                        $open_count_prev = $row[2];
                                }
                                $now = $row[0];
                                $open_count_now = $row[2];
                                $first_minus_sent_now = $row[1];
                                if(strcmp($prev, $now) != 0){
                                        $open_activity_array[0] = $prev;
                                        $only_email[0] = $prev;
                                        if($overall_count==1){
                                                $open_activity_array[1] = $count;
                                        }
                                        else{
                                                $open_activity_array[1] = $count+1;
                                        }
                                        $open_activity_array[2] = $open_count_prev;
                                        $open_activity_array[3] = 
$first_minus_sent_prev;
                                        fputcsv($open_file_inactivity, $only_email);
                                        fputcsv($count_file, $open_activity_array);
                                        $prev = $now;
                                        $open_count_prev = $open_count_now;
                                        $first_minus_sent_prev = 
$first_minus_sent_now;
                                        $count = 0;
                                }
                                else{

if($first_minus_sent_now<$first_minus_sent_prev){
                                                $first_minus_sent_prev = 
$first_minus_sent_now;
                                        }
                                        $count = $count + 1;
                                        if($overall_count==0){
                                                $overall_count = $overall_count+1;
                                        }
                                        else{
                                                $open_count_prev+=$open_count_now;
                                        }
                                }
                        }
```

Care should be taken to write the first record properly. The algorithm previously stated will not work for the first row as there is no 'prev' record for the first row. Hence for the first condition, the prev and the now records are equated and a separate if-condition is written at the end of the main loop. overall_count variable is used to maintain this condition – if it is zero, then do nothing to the total_opens count. If it is non zero, only then start adding the total_opens values.

An open_data_inactivity file is also maintained that will act as a tool to isolate the inactive email IDs from the sent email list (those email IDs that are not there in open_activity_data but still have received a campaign are inactive). This is the reason why an array only_email id maintained in the loop and is written to a separate file pointer `$open_file_inactivity.`

Again it is necessary to note that the exit condition is reached when the end of the file is reached. The end of the file is the last record, so the loop exits before the last record is written. Hence after the loop is completed it is necessary to write the last record (still in the array) to the csv file.

click_collect.php

open_collect.php is a class file **Load_Click** that consists of three functions – put_sort_csv(), reduce_to_count() and reduce_to_count_auto(). put_sort_csv() is used to sort the data and put it into the csv file for the reduce function in manual mode. reduce_to_count() is the reduction function for manual mode and reduce_to_count_auto() is the reduction function for auto mode.

The reduce function uses the same logic as that for sent_collect since we really need only the count of the email ID along with the email ID itself. In the program submitted, there is a provision to extract the click time minus campaign sent time by accessing `$row[1]`.

ultimate_reduce.php

This is the final PHP script – a class file **Reduce** that will run in the Reducing Redundancies in the existing data step. This program uses up to five file pointers – three for reading (one each from sent, open and click respectively) and two for writing (one for writing the reduce data and one for writing the abnormal data).

Defining abnormal Data – There are some entries in the open_activity_table that will distort the classification results. Some of the best examples is when an entry is present in the open_activity_table that has opened a campaign after a month of sending the campaign. The usual life time of a promotional campaign will be one month on an average and for a non-promotional campaign not be more than a week. Hence such campaigns that are opened after this time are considered to be abnormal and are written separately in abnormality.csv file.

Similarly in the open_activity_table data, entries that have more than a particular amount of opens are also regarded to be abnormal and they distort the classification. A general limit for this is around 10, but in some cases it could be more. But generally an email ID opens a campaign only once, and having an open count more than 10 would be certainly abnormal and is probably the result of automated scripts on the email ID's side.

This limit can be changed in the config.php file and is controlled by the DAYS_LIMIT and OPEN_LIMIT constants.

Algorithm –

1. Compare the entries in Reduced Open Data (open_data_reduced.csv) and Reduced Sent Data (sent_data_reduced.csv). If they both match then proceed to Step 3. Else proceed to Step 2.
2. Keep going to the next pointer in the sent_data_reduced.csv till there is a match using fgetcsv (fgetcsv automatically goes to the next line every time it is called).
3. If there is a match between open_data_reduced row and sent_data_reduced row's email IDs then look for the corresponding file pointer in the click_data_reduced.csv file – if there is a match, Proceed to Step 5. Else proceed to Step 4.

4. If there is no match in Step 3, then that particular email ID does not have an entry in the click_activity_table. Hence the number of clicks are taken to be zero (0). The remaining features are calculated and stored in `$reduce_array.`
5. If there is a match in Step 3, then take the corresponding count in Click Data and find the click through rate for the feature. Only then move to the next file pointer using the filepointer->next() function.

The code snippet is as follows:

```
while(!feof($open_data_file) || !feof($sent_data_file)){

          if(feof($open_data_file) || feof($sent_data_file))

                 break;

          $row_open = fgetcsv($open_data_file);

          if($save_count = 0 && strcmp($row_open[0], $row_sent[0])==0){

              if(strcmp($row_open[0], $row_click[0])==0){

                    $reduce_array[0] = $row_open[0];

                    $reduce_array[1] = (float)$row_open[2]/(float)$row_sent[1];

                    $reduce_array[2] = (float)$row_open[1]/(float)$row_sent[1];

                    $reduce_array[3] = (float)$row_click[1]/(float)$row_sent[1];

                    $reduce_array[4] = $row_open[3];

                  if($reduce_array[4]/86400<=DAYS_LIMIT && $row_open[1]<=OPEN_LIMIT){

                          fputcsv($reduce_data_file, $reduce_array);

                  }

                  else{

                      fputcsv($abnormal_file, $reduce_array);

                  }

                  $click_data_file->next();

                  }

           else{

                $reduce_array[0] = $row_open[0];

                $reduce_array[1] = (float)$row_open[2]/(float)$row_sent[1];

                $reduce_array[2] = (float)$row_open[1]/(float)$row_sent[1];

                $reduce_array[3] = 0;

                $reduce_array[4] = $row_open[3];

                if($reduce_array[4]/86400<=DAYS_LIMIT && $row_open[1]<=OPEN_LIMIT){
```

```
            fputcsv($reduce_data_file, $reduce_array);
        }
        else{
             fputcsv($abnormal_file, $reduce_array);
        }
      }
      $save_count = $save_count + 1;
    }

    while(strcmp($row_open[0], $row_sent[0])!=0){
         $row_sent = fgetcsv($sent_data_file);
         if((float)$row_sent[1]!=0.0){
            if(strcmp($row_open[0], $row_sent[0])==0){
            $row_click = str_getcsv($click_data_file->current());
            if(strcmp($row_open[0], $row_click[0])==0){
                $reduce_array[0] = $row_open[0];
                $reduce_array[1] = (float)$row_open[2]/(float)$row_sent[1];
                $reduce_array[2] = (float)$row_open[1]/(float)$row_sent[1];
                $reduce_array[3] = (float)$row_click[1]/(float)$row_open[2];
                $reduce_array[4] = $row_open[3];
        if($reduce_array[4]/86400<=DAYS_LIMIT && $row_open[1]<=OPEN_LIMIT){
              fputcsv($reduce_data_file, $reduce_array);
         }
        else{
             fputcsv($abnormal_file, $reduce_array);
        }
        $click_data_file->next();
      }
      else{
            $reduce_array[0] = $row_open[0];
            $reduce_array[1] = (float)$row_open[2]/(float)$row_sent[1];
```

```
                    $reduce_array[2] = (float)$row_open[1]/(float)$row_sent[1];
                    $reduce_array[3] = 0;
                    $reduce_array[4] = $row_open[3];
               if($reduce_array[4]/86400<=DAYS_LIMIT && $row_open[1]<=OPEN_LIMIT){
                     fputcsv($reduce_data_file, $reduce_array);
               }
               else{
                    fputcsv($abnormal_file, $reduce_array);
               }
              }
            }
        }
     }//end of while loop for matching sent and open emails
 }   //end of main loop
```

File pointer that is used for processing click_data_reduced.csv file is of PHP SplFileObject() type. When the object is define this way there is a function $file_pointer->next() that is available to manually move the file pointer. Thus our final required output is present in the reduce_data.csv file marked with the appropriate timestamp the program had started.

The files that are in the temp directory are deleted with PHP commands unlink and rmdir.

Temp Files

The above programs create many temporary files and many permanent files. All of these files will have a timestamp (TIMESTAMP) before their file names. They include:

1. sent_data.csv – Ordered output of sorting from MySQL.
2. sent_data_reduced.csv – Reduced Data from sent_data.csv that consists of unique sent email IDs with the number of times they occur as a comma separated field along with the email ID itself.
3. open_activity_data.csv – Ordered output of sorting and joining from MySQL.
4. open_data_reduced.csv - Reduced Data from open_activity_data.csv that consists of unique sent email IDs with the following fields that appear as comma separated values:
   i. sum of all total_opens
   ii. unique opens
   iii. first open time minus campaign sent time
5. click_actvity_data.csv – Ordered output of sorting and joining from MySQL.
6. click_data_reduced.csv - Reduced Data from sent_data.csv that consists of unique sent email IDs with the number of times they occur as a comma separated field along with the email ID itself.

Classification of the email IDs based on their activity

The problem of classification based on email IDs activity can be viewed as an unsupervised learning classification problem as it involves ordering data that are close together in values. In case of promotional campaigns having data for 15 or more days will give a more accurate and versatile result.

K-Means clustering was used to get the classification result. It is a type of cluster analysis.
Cluster analysis or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters).

K-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function known as squared error function given by:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\left\| x_i - v_j \right\| \right)^2$$

Where,

'$||x_i - v_j||$' is the Euclidean distance between x_i and v_j.

'c_i' is the number of data points in i^{th} cluster.

'c' is the number of cluster centers.

Algorithm for K-Means

Let X = $\{x_1, x_2, x_3 \dots x_n\}$ be the set of data points and V = $\{v_1, v_2 \dots v_c\}$ be the set of centers.

1. Randomly select 'c' cluster centers.

2. Calculate the distance between each data point and cluster centers.

3. Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.

4. Recalculate the new cluster center using:

$$\boldsymbol{v}_i = (1/c_i) \sum_{j=1}^{c_i} \boldsymbol{x}_i$$

Where, ‘c_i’ represents the number of data points in ith cluster.

5. Recalculate the distance between each data point and new obtained cluster centers.

6. If no data point was reassigned then stop, otherwise repeat step 3

Scaling the data – The features available are not all directly proportional to the measure of activity. The first open time minus campaign sent time is a measure of how quickly the email ID opens the campaign sent to it. However if the number is large, then it is relatively less active. When comparing this situation with the other features, more open rate or click through rate means more the activity. Hence the feature of first open time minus campaign sent time needs to be reversed and this is done by the following formula:

$$first\ minus\ sent = \frac{\max first\ minus\ sent - first\ minus\ sent}{avg\ first\ minus\ sent}$$

Flow of classify.py - The module used for K-Means clustering is SciKit Learning (import sklearn). It has an inbuilt library for performing K-Means. First the data is normalized using the whiten function in the SciKit K-Means library. Then the fit function is used for the actual classification process. Once the fit function is complete the labels function is called which returns an array of the appropriate labels. As a final step, the list of email IDs that is stored in an array is matched with the list of labels.

Input File – For getting the location of the TIMESTAMP.”-reduce_data.csv” file since the folder name is randomly generated, the file location is written in a subdirectory /support files with the file name of the following format: for_python_USER_ID.txt which will be read inside classify.py file during run time.

The user id is picked up by the python program from the input argument using `sys.argv[1].` **However** for the PHP scripts it is required to **change the user id in config.php file under the constant USER_ID.**

Exit condition – In case of a realistic data set, the following holds true:

$$\boldsymbol{Normal > Active > Superactive}$$

Hence the exit condition when the number of Active and Superactive users differ by a large amount is undoubtedly the above. However when this is not the case

Tolerance – In case of a dataset where there are more number of Superactive users than Active users, there is usually very less difference between the two numbers. Hence it was seen that the labels get interchanged if the exit condition is going to be

$$\boldsymbol{Normal > Active > Superactive}$$

Instead it should be

$$Normal > Superactive > Active$$

Looking at the data and observing where the max click through rate lies, that should get classified as Superactive and hence the maximum click through rate is stored (which is usually 1) and the label assigned to this maximum click through rate and interchanged with that of the Superactive label.

A tolerance percentage difference between Superactive and Active labels determines this which can be changed in classify.py – where the variable is named as **tol.**

Using config.php

The config.php file will contain three sets of variables:

1. Directories
2. Procedural Flow variables
3. Database variables

The directories are the follows:

1. INPUT_SENT_PATH – should be in the format of DIR/input/sent with MySQL read/write permission given for the same.
2. INPUT_OPEN_PATH – should be in the format of DIR/input/open with MySQL read/write permission given for the same.
3. INPUT_CLICK_PATH – should be in the format of DIR/input/click with MySQL read/write permission given for the same.
4. OUTPUT_PATH – where the output files will be present. Inside this a temporary folder temp will be created that will contain intermediary files and will be deleted once the program finishes executing. If this directory does not exist, the program itself will create it with a Random Key generated and ending with the USER_ID of the client.
5. CAMPAIGN_TABLE_DIR – campaign_table data needs to be loaded if it is not already present. Else an error will occur in both open_collect.php and click_collect.php wherever `$open_data_file` and `$click_data_file` variables are used.

The following are the Procedural Flow variables:

1. DELETE_EXISTING_DATA – To truncate the existing table corresponding to the USERID. (Y/N)
2. LOAD_DATA – To load open_activity_table and click_activity_table data. (Y/N)
3. LOAD_DATA_SENT – To load campaign_sent_table data. (Y/N)
4. LOAD_CAMPAIGN_DATA – To load campaign_table data. (Y/N)
5. SORT_SENT_DATA – To sort sent data and put in a csv file. Change to N if file already present. (Y/N)
6. SORT_OPEN_DATA – To sort open data and put in a csv file. Change to N if file already present. (Y/N)
7. SORT_CLICK_DATA – To sort click data and put in a csv file. Change to N if file already present. (Y/N)
8. DAYS_LIMIT – Deciding the days limit for first open minus campaign sent time abnormality.
9. OPENS_LIMIT – Deciding the opens limit for total_opens.

The following are the Database variables:

1. DATABASE – database name
2. HOST – hostname
3. USERNAME
4. PASSWORD
5. OPEN_ACTIVITY_TABLE
6. CLICK_ACTIVITY_TABLE
7. CAMPAIGN_SENT_TABLE
8. CAMPAIGN_TABLE

Assumptions of the config.php

1. It is assumed that the campaign table information (with campaign sent time for a particular campaign ID is already present in the campaign table) if the LOAD_CAMPAIGN_DATA constant is declared with N. Hence before running the script it is necessary to check if LOAD_CAMPAIGN_DATA is given Y/N depending on the already present data. For a client if a new list (.csv file) is used for this purpose, give the corresponding file name WITH the file directory in CAMPAIGN_TABLE_DIR with MySQL read/write permission given using chown command in LINUX.
2. Make sure the input directory is given MySQL read/write permission using chown command in LINUX.

Running the Program

A bash file (run.sh) is created that consists of calling the two programs (test.php and classify.py) procedurally. While running the bash file for a user **after** making the necessary changes in the config.php, it is necessary to give the **name of the config.php** file as an argument for the bash file. For example if the config file is named as **config_1234.php and** user id is **1234** then for running the program in the command line give:

```
>sh run.sh config_1234 1234
```

Caveat: The name of the config.php file and user id is automatically picked up and run by the program. The user id is for classify.py file and test.php file.