BIRKBECK, UNIVERSITY OF LONDON

MSc Cognitive and Decision Sciences

How much behavioural targeting can help online display advertising?

Supervisor: Professor of Behavioural Science, Nick Chater

Candidate: Radek Maciaszek

Date: September 2010

Contents

Abstract

Online advertising has exploded during the past few years; the current UK market is evaluated at £3.5 billion, and it grew dramatically by about 2200% during 2000s[1]. Behavioural targeting (BT) is largely regarded as one of the most effective technique in optimizing online advertising. However, despite the impressive numbers involved in this industry, there are only a few academic studies performed on real world click-stream data. (e.g. Yan, Liu, Wang, Zhang, Jiang & Chen 2009; Ratnaparkhi 2010; Chen, Pavlov, & Canny 2009). This may be linked to the extreme demands on system resources required by the massive amount of advertising data available. Yan et al. (2009) confirmed that BT can significantly increase the effectiveness of one specific type of online advertising, the so-called search advertising. In this work we investigate whether techniques linked to BT may be beneficial to online display advertising. Using data from a major commercial ad network, we show that a simple BT technique such as user clustering could improve click-through ratio by more than 900%.

Furthermore, from a software engineering perspective, we provide support for using distributed open source technologies to tackle the complex analysis of advertising data.

1. Introduction

The history of modern advertising in mass media started in 1630s, when Théophraste Renaudot printed the first advertising in the French newspaper La Gazette de France (Pincas & Loiseau 2008). The first online advertising appeared much later in 1994 with the creation of Netscape browser. Growing popularity of Internet and fast adoption of Internet browser as a communication channel started a trend of migrating advertising revenue from traditional media towards the Internet. According to Barclays Capital the U.S. online ad revenue in 2009 reached

[1] Source Internet Advertising Bureau UK

almost 10% of the U.S. spending on advertising summing up to over $240 billion. The increase in the penetration of Internet which reached in 2009 over 77% of the entire U.S. population and 28% worldwide[2] made the Internet an important advertising mediums, on equal footing with traditional print, TV and radio advertising.

Perhaps one of the most important features of Internet advertising is that it stretches the traditional definition of advertising as a one way mass-media communication channel. Internet allows advertisers to communicate with users instantly by presenting a content which users can interact with. Marketers can reach individual users based on their actions or their geographical location. The interactive nature of Internet allows marketers to provide instant feedback to the marketing campaign by recording how end users are interacting with the ads.

Internet advertising can be divided into separate business models such as search and display advertising. Search advertising is associated with ads showed along the results returned by a search engine and typically it contains text only ads. Display advertising usually consists of static or dynamic images; it appears on web pages and is often used for branding (Dreze & Husherr 2003). One advantage of search over display advertising is that it is easier to understand the interests of users who provide it directly as search keywords. Understanding interests of visitors of websites who see display ads is a more subtle task and requires use of such techniques such as cookies or targeting based on user geographical location.

Effectiveness of advertising campaigns is playing increasingly important role in online advertising. There are many metrics by which the advertising campaign can be judged and many ways in which it can be improved. In the ideal situation the advertising effects can be measured by the increase in sales caused by online campaign. Often however such a measurement is not possible, especially when online campaign advertises products which cannot be bought on the Internet. The effectiveness of advertising depends in the first place on the objectives which were

[2] Source Internet World Stats: http://www.internetworldstats.com/stats.htm

set for the advertising campaign. Sometimes the role of advertising is to build a brand image or simply to inform about new product which only indirectly can lead to higher sales. In this work the main metric used to evaluate the improvement in delivering the ads will be a click-through ratio (CTR). CTR is the number of times the ad was clicked divided by the number of times the ad was presented to all users.

The basic strategy of increasing the effectiveness of ads is based on targeting, that is controlling which ads are presented to which users. This process can be achieved in many ways. Examples include such techniques as geographical targeting which allows for targeting ads based on physical location of Internet users. It is also possible to target ads based on time or by the context of the page user visits. For example an advertiser may place car ads on a website about sport cars. There are dozens of parameters which can be utilised in the ad optimisation and targeting. However, in this project we will focus on targeting based only on one parameter which will define similarity between Internet users. We will assume that users who visited same websites are more similar to each other than users who visited different websites and that similar users are more likely to click on the same ads. This assumption will be tested using real data recorded by one of the biggest online ad networks in United Kingdom.

One of the intrinsic problems in performing real life samples is gigantic volume of data which needs to be analysed in order to find answers to even the most straightforward questions. The amount of traffic recorded by big ad networks often amounts to billions of impressions (ad views) a month and the file sizes involved in calculations easily go into terabytes. The large number of statistical data renders most of the standard statistical tools unusable and requires new approaches. In the case of this project just the data from one day requires analysis on millions of vectors with over one hundred thousand dimensions each. Because of this single requirement the method of analysing such immense amount of data is one of the most important aspects of this project.

The goal of this work is to examine whether BT techniques shown to improve effectiveness of search advertising (Yan et al. 2009) may be applied successfully to online display advertising. A special technique specific to BT (user clustering) will be applied to data recorded by a commercial ad network during one day on 4th of August 2010.

2. Online advertising and behavioural targeting

Ad servers record both ad views (so called impressions) and clicks. More advanced ad servers are able to gather additional information about each user such as exact time when user interacted with the ad, her/his geographic location, the Internet connection speed, resolution of the screen or the type of browsers used and many more parameters. Online ad networks are becoming increasingly sophisticated in using wide range of parameters passed by users' browser to improve the effectiveness of online advertising. It has been shown that online advertising has different effects based on user's gender (Wolin 2003) or nationality (Brettel 2010). Since it is possible to instruct ad server to behave differently based on for example user's gender then trivially we can assume that behavioural targeting should work in those cases. However, usually it is not possible to get personal information about Internet users and different means of distinguishing between Internet users are necessary.

This research will differentiate between users based on their click-stream activity, that is based on all pages visited by and all clicks resulted from users actions while seeing ads. Click-stream activity was recorded by the ad server and is stored in our dataset. There are subtle differences between this approach and the research done by Yan et al. (2009). First of all in the research by Yan et al. users are clustered based on the URLs they clicked in the search engine. In our case we will use URLs of the pages which users visited as opposed to the URLs of ads on which users clicked. Typically the average CTR in display advertising is very low (in our dataset it is below 0.1%). Therefore, it is better to use URLs visited by users to build their profiles rather than the URLs from the ads, otherwise we will lack the data for over 99.9% of all users. Additionally in order to minimise the number of dimensions in each user's vector we only use the domain part of each URL. Thus if a user visited two URLs "www.example.com/page1.html" and "www.example.com/page2.html" both of those URLs are treated as the same dimension "www.example.com". We use the words "domain" and "URL" interchangeably since all the URLs which are analysed in this research are represented by their domain only.

3. Distributed processing

Most of the computations which need to be performed in this research are conceptually straightforward. However, the input data is substantially large and the computations have to be distributed across many servers which makes this task non-trivial. One way of dealing with distributed data processing is offered by MapReduce programming model (Dean & Ghemawat 2008) which was originally developed by Google and is available in several Open Source implementations. This project will use the Apache Hadoop implementation of MapReduce framework which is available as Open Source software.

MapReduce is a programming model which simplifies parallelisation of computations and allows researcher to focus on implementing the specific problem which needs to be calculated. MapReduce automatically parallelize the implemented algorithms and hides from us all additional details required for parallelisation such as fault tolerance, data distribution and load balancing. A program written in a MapReduce framework may be easily executed on several machines thus allowing for analysis on large body of data.

A more natural way of analysing data is offered by SQL-like database which allows asking questions in a SQL-like language. Using SQL is usually a more convenient way of analysing data than implementing the same logic in one of the standard programming languages. The same applies to MapReduce as implementing algorithms in MapReduce framework is typically more complicated than using SQL queries. This is the reason why this project uses Apache Hive.

Hive is a datawarehouse database which works on top of Hadoop and translates SQL commands into underlying MapReduce procedures which are executed by Hadoop.

Hadoop and Hive make processing of large quantities of data relatively easy. However, data mining algorithms such as K-means, Cluto or Canopy (Kanungo et al. 2000) require separate implementation. This project used Apache Mahout project which includes K-means algorithm implemented in a MapReduce Hadoop framework. Mahout is a scalable machine learning library which among many others implements various clustering algorithms. Mahout allows to distribute the computing on many servers and in this way it speeds up the total calculations time.

After selecting the technology required to implement required calculations it was necessary to find flexible cloud hosting with the support for Hadoop. While it is possible to install Hadoop on almost any server the preferred solution would be not to acquire any hardware. This project used Amazon Elastic MapReduce – a flexible on demand computing in the cloud, which offers support for Hadoop and works on top of Amazon Elastic Compute Cloud (EC2).

Just how important was the use of MapReduce shows the total time for running all algorithms. Overall all analysis took over 4000 normalized hours, that is how much time it would take if all analysis would be performed on one small Amazon EC2 server (so called m1.small). In other words it would take over 166 days to perform analysis on a single machine. Thanks to cloud computations this time was compressed into just few weeks.

4. Dataset

The data analysed in this experiment was recorded by one of the leading UK ad networks (which asked to keep their name confidential). All data was recorded during one day on August 4th 2010. The dataset contains two types of users' actions – impressions and clicks. Every time a person saw an ad on any of the websites within ad network that fact was recorded in database as an "impression". Each time someone clicked on an ad that fact was recorded in database as a "click". The dataset is rather large as it contains over 80 millions of impressions records and over 60 thousands of clicks just within single day of serving online advertising. Overall in a single day over ten millions of unique users visited over one hundred and forty thousand different websites (counted by domains) and saw almost two thousands ads. For a comparison the study published by Yan et al. (2009) analysed 6 millions of records recorded during 7 days.

We filter all records to ensure that internet robots hits are not included in our dataset. Internet bots, such as web spiders, are software applications that browse WWW usually in order to provide data for search engines. To filter out all robots actions all users who recorded more than 100 clicks or impressions are removed from the dataset. To avoid any privacy concerns we will not store or analyse any private user information that is any information which can be used to identify users – such as users IP or their exact geographic location.

The format of both clicks and impressions is exactly the same and is presented in table 1. The data is saved in hourly data files which are stored in Amazon Simple Storage Service (S3). Amazon S3 provides a flexible storage where gigabytes of source data are securely stored and can be easily accessed by Amazon EC2 cloud computing servers.

Name	Sample Data	Description
Event Time	2010-08-01 12:30:04	The time when a user saw an ad
User ID	546c14241e0f1aa5a0e54420b44f4e2f	Unique user ID which is stored in user browser as a cookie and can be

		used to identify a user.
Ad ID	60041	Unique Id of each ad
Page URL	http://www.example.com/page.html	Page where the ad was presented to user.
Referrer URL	http://www.google.com	Page which user arrived from before he saw an ad.
Campaign ID	12243	ID of the campaign. Each campaign groups one or more ads which typically advertise the same product for the same advertiser.
Advertiser ID	398	ID of the advertiser. Each campaign belongs to one and only one advertiser. Typically campaigns are thematically similar as they advertise products from the same advertiser.

Table 1. Format of impressions and clicks log used in our experiment

4.1 Data representation

In order to perform data mining analysis on our dataset we need a way to represent it in the numerical format. One way of representing URLs is to count the frequency of occurrences of each term (in our case a domain part of each URL) and represent users vectors using term frequency weighting. An improved form of this representation is a term frequency – inverse document frequency (TF-IDF) weighting (Salton & Buckley 1988; Papineni 2001).

TF-IDF weight is commonly used in text and data mining. This statistic measure is used to evaluate how important is a given term in the corpus of all documents. Term frequency shows how many times a given term occurred in a given document and an inverse document frequency shows the importance of a term by dividing number of documents containing a given term by total number of documents. Inverse document frequency allows us to give less weight to very common words.

Each user (u_{ij}) is represented as a vector of TF-IDF weights:

$$u_{ij} = (\log(a) + 1) \times log\frac{l}{b}$$

Where:

- $i = 1,2, \dots$f indicates each user, f is the number of all users
- $j = 1,2, \dots l$ for each URL, where l is the number of all URLs
- a is a number of times user i visited URL j
- b is a number of all users who visited URL j

Intuitively we can see that if the URL occurs frequently then the document frequency is large and the inverse document frequency will be small. The inverse document frequency is normalised with the number of terms. Finally the logarithm is used in order to decrease the effect of term frequency on the final weight.

Representing users as vectors of TF-IDF weights has some computational drawbacks as each of the over 10 millions of users consists of a vector containing over one million of weights, one per each URL. This brings some computation challenges and as a result some further simplifications are required. One way to minimize the number of dimensions is to group similar URLs by their domain. This optimisation decreased the number of dimensions to over one hundred thousand per every vector.

The side effects of representing users as vectors containing TF-IDF weights is that most of users did not visit large majority of the URLs and therefore most of weights in each user vector equal zero. In order to minimize the number of dimensions kept in each vector we represent those using sparse vectors. Sparse vector stores only non-zero values assuming that all other values are equal zero. It is an often case in data mining that vectors have a large number of dimensions where most of them are zeros and Apache Mahout supports this data representation.

5. Experimental Design

To ensure that results of our experiment can be compared with the results achieved in paper publisher by Yan et al. (2009) we tried to keep the symbols and experimental setup as similar as possible. Some things needed to be changed since there are few major differences between the data being analysed in this and the research by Yan et al. (2009). One major difference is that the experiment is performed on display advertising as opposed to search advertising. Another difference is that experimental data is recorded during 24 hours of serving ads as opposed to 1 and 7 days. Running time of programs became so significant that analysing more that one day worth of data was not feasible.

5.1 Symbols

This section contains definitions of mathematical symbols which are used across experiment. The set of n advertisements is represented as:

$A = \{a_1, a_2, \dots, a_n\}$

Let $U_i = \{u_{i1}, u_{i2}, \dots u_{im_i}\}$ be a set of such users who displayed or clicked on ad a_i. The dataset (see table 1) contains UserID and Ad ID which uniquely identify each user and are used to count how many times each user saw each ad.

A boolean function $\delta(u_{ij})$ defines if a user u_{ij} clicked on the ad a_i:

$$\delta(u_{ij}) = \begin{cases} 1 \ if \ u_{ij} clicked \ a_i \\ 0 \ otherwise \end{cases}$$

The main goal of a behavioural targeting strategy is to group users into separate clusters which allow to simulate delivery of different ads to different groups of users. A distribution of such n users into K clusters is defined as a function:

$$G(u_i) = \{g_1(U_i), g_2(U_i), \dots, g_K(U_i)\}, where \ i = 1,2, \dots n$$

Each $g_k(U_i)$ indicates all users from users set U_i who were grouped into the kth clustering subset. Such a kth user segment can be represented as:

$$g_k = \bigcup_{i=1,2,\dots n} g_k(U_i)$$

5.2 Evaluation metrics

Some of the evaluation metrics used in this experiment are common in online advertising, such as CTR and overall CTR improvement. Additionally this research will use some of the statistics used by Yan et al. (2009), that is within- and between-ads similarity, precision, recall and F-measure. Finally we will use t-test to confirm the significance of our results.

5.2.1 Within- and Between- Ads similarity

The within- and between-ads metrics attempt to answer a question whether users who clicked the same ads are more similar to each other than to users who clicked different ads. This statement is a basic assumption of any behavioural targeting technique.

We define the similarity between two user vectors in terms of classic Cosine similarity:

$$Sim(u_{ij}, u_{st}) = \frac{< u_{ij}, u_{st} >}{||u_{ij}||||u_{st}||}$$

where <,> is a vector inner-product and || is a vector 2-norm. Since TF-IDF weights cannot be negative this metric should give us values from the range 0 to 1.

Given the above Cosine similarity measure it is possible to define the within- and between similarities metrics. The within ad similarity is an averaged sum of Cosine similarity of all users who clicked given ad a_i. For each ad a_i we define a within-ads user similarity as:

$$S_w(a_i) = \frac{2}{l_i(l_i - 1)} \sum_{\partial(u_{ij})=1} \sum_{\substack{\partial(u_{it})=1 \\ t \neq 1}} Sim(u_{ij}, u_{st})$$

where $l_i = \sum_j \partial(u_{ij})$ is a number of users who clicked ad a_i.

Between ads similarity measure resembles the within-ads similarity definition with the difference that it measures the similarity between users who clicked different ads. The between ads similarity answers the question how similar are users who clicked different ads. Between ads similarity is defined as:

$$S_b(a_i, a_s) = \frac{1}{l_i l_s} \sum_{\partial(u_{ij})=1} \sum_{\partial(u_{st})=1} Sim(u_{ij}, u_{st})$$

Where:

- $l_i = \sum_j \partial(u_{ij})$ is a number of users who clicked ad a_i
- $l_s = \sum_j \partial(u_{sj})$ is number of users who clicked ad a_{ji}.

Finally, using within and between ads similarity we define a ratio between $S_w(a_i)$ and $S_b(a_i, a_s)$ as:

$$R(a_i, a_s) = \frac{S_w(a_i) + S_w(a_s)}{2S_b(a_i, a_s) + d}$$

The R measure will increase with the increase of within-ads similarity and decrease if the between-ads will be bigger. Therefore the bigger the R measure the more likely that our behavioural targeting strategy achieved its desired results.

It is possible that the similarity between some ads (S_b) may equal 0. In those rare cases we may not be able to calculate the R measure. We introduced therefore a new parameter which holds a very small value (d = 0.001) just to ensure that the division in our case is always possible. It is worth noting that Yan et al. (2009) do not mention any issues with calculating R in all cases.

Finally we use S_w, S_b and R to evaluate how similar are all ads within the dataset. In order to do so we will calculate the average within and between ad similarity:

$$S_w = \sum_i \frac{S_w(a_i)}{n} \quad \text{and} \quad S_b = \sum_i \frac{S_b(a_i)}{n}$$

The total averaged ratio R is calculated as an squared average of all ads ratios R:

$$R = \sum_i \sum_s \frac{R(a_i, a_s)}{n^2}$$

5.2.2 Ads Click-Through Rate

The CTR measure is the most common performance indicator of any click-based advertising campaign. Typically a high CTR ratio indicates that users are more interested in a given ad and are paying more attention to advertising message (Joachims et al. 2005). Many advertising campaigns use more sophisticated indicators such as measures of whether Internet users “converted” into customers. That is if a user after clicking or seeing an ad acted in a specific way, for example whether s/he bought the product being advertised.

We define CTR ratio of an ad as a number of users who clicked the ad divided by number of users who clicked or displayed it:

$$CTR(a_i) = \frac{1}{m_i}\sum_{j=1}^{m_i} \partial(u_{ij})$$

To see if the user segmentation can increase an ad CTR we define the CTR of ad a_i over user cluster g_k as:

$$CTR(a_i|g_k) = \frac{1}{|g_k(U_i)|}\sum_{u_{ij}\in g_k(U_i)} \partial(u_{ij})$$

where $|g_k(U_i)|$ is the number of users in kth cluster $g_k(U_i)$. After segmenting users in k clusters we check the CTR of each ad per each cluster to see if delivering the ad only to given cluster would improve its CTR.

After calculating the CTR for each ad-cluster pair we answer the question just how much the overall CTR can be increased by targeting users to different user groups. This can be done by taking the maximum CTR of each ad from all clusters we can possibly deliver it to and averaging the resulting sum:

$$\Delta(a_i) = \frac{CTR\big(a_i|g_*(a_i)\big) - CTR(a_i)\big)}{CTR(a_i)}$$

where $g_*(a_i)$ is user segment which has a highest CTR for a given ad a_i:

$$g_*(a_i) = argmax\{CTR(a_i|g_k), k = 1,2, \dots , K\}$$

It is important to note that it is not guaranteed that the user segment with the highest CTR for a given ad a_i has the biggest number of impressions for that ad. We will discuss implication of this assumption later in the discussion section.

5.2.3 Precision, Recall and F-measure

There are few ways in which we can assess the quality of CTR improvement. Calculating precision and recall is a common way to check the quality of captured data (Rijsbergen 1979). If we consider users who clicked on ads as positive cases and users who saw the ad but did not click on it as a negative case we can calculate the precision as:

$$Pre(a_i|g_k) = CTR(a_i|g_k)$$

The precision in this case is equal to the number of users from given cluster g_k who clicked the ad divided by the number of all users (from cluster g_k) who saw given ad which is the CTR of an ad in given user segment.

Recall tells us how much percent of all users who clicked an ad are within given user segment. It is defined as a number of users from cluster g_k who clicked an ad divided by the number of all users from an entire dataset who clicked the same ad:

$$Rec(a_i|g_k) = \frac{\sum_{u_{ij}\in g_k(U_i)} \partial(u_{ij})}{\sum_{j=1}^{m_i} \partial(u_{ij})}$$

where m_i is the number of users in our dataset.

A high precision will indicate that segments are tightly clustered around users who clicked given ads while recall will show us how much of all the clicks on a given ad where included in a given cluster.

We can further combine both of these measures into a harmonic mean of precision and recall called F-measure:

$$F(a_i|g_k) = 2 \times \frac{Pre(a_i|g_k) \times Rec(a_i|g_k)}{Pre(a_i|g_k) + Rec(a_i|g_k)}$$

The F-measure (Hripcsak & Rothschild 2005) shows how well our clustering is performing for a given ad a_i and a cluster g_k.

The F-measures from all clusters and ads can be summed into a total F measure averaged for all ads and best performing clusters:

$$F = \sum_{i=1}^{n} \frac{F(a_i|g_*(a_i))}{n}$$

6. Implementation details

Large datasets are common in Internet advertising yet it is still a difficult task to perform data mining on data which contains many billions of records. A researcher who is analysing such datasets needs to use scalable technology in order to process such quantities of data. Most of the technologies used to implement data mining algorithms in this project are focused on distributed processing using MapReduce framework.

The programming languages of choice are Python for text processing and preparation of input vectors, Hive SQL for high level data transformations and Java for implementing k-means clustering.

K-means clustering is a standard clustering method (Kanungo et al. 2000) which can be used for clustering numerical vectors. K-means partitions vectors into k clusters by assigning each vector to the closest cluster centre. Initial cluster centres can be selected randomly in the first iteration of the algorithm or can be provided to Mahout as a separate set of vectors. The k-means algorithm calculates new “means” vectors by calculating a centroid within each cluster and in next iteration it repeats a procedure by assigning each vector to new cluster centers.

6.1 Servers setup

All computations were executed on Amazon Elastic Compute Cloud (EC2) which provides flexibility in choosing servers size and can dynamically allocate time required to execute all experiments.

Data mining analysis were performed on five high-CPU instances with the following configuration:

7 GB of memory

20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each)

1690 GB of storage

64-bit platform

All servers are based on Debian Linux and have installed Hadoop v 0.20.

6.1 Dataset

The data used in the experiment is stored in the tab separated format which needed to be imported into Hive database for further analysis. Hive represents data in easy to use table format, see listings 1 and 2. Note that both clicks and impressions have the same format.

```
CREATE EXTERNAL TABLE event_clicks (
    event_time string,
    user_id string,
    banner_id int,
    page_url string,
    referrer_url string,
    campaign_id int,
    advertiser_id int
 )
 PARTITIONED BY(event_date string)
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';
```

Listing 1. Clicks Hive table definition

```
CREATE EXTERNAL TABLE event_impressions (
    event_time string,
    user_id string,
    banner_id int,
    page_url string,
    referrer_url string,
    campaign_id int,
    advertiser_id int
 )
 PARTITIONED BY(event_date string)
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';
```

Listing 2. Impressions Hive table definition

6.2 Building user vector

To create users' vectors we need to calculate the TF-IDF weights for each user-domain pair. In order to do so we need to create a function which retrieves the domain name from every URL.

```
import sys
from urlparse import urlparse

def getDomain():
    for line in sys.stdin:
        line = line.strip()
        if line == "":
            continue
        (components) = line.split("¥t")
        url = components[0]
        parts = urlparse(url)
        output = parts.netloc
        if len(components) > 1:
            isFirst = True
            for component in components:
              if isFirst:
                  isFirst = False
                  continue
```

```
                output += '¥t' + component
            print output

if __name__ == "__main__":
    getDomain()
```

Listing 3. Python file domain.py used for transforming URLs to domains

We create first a helper table which contains user Ids, domain names and number of times each user have seen any ad under given domain. Hive does not offer a function which can retrieve domain components from the URL names and in order to do so a custom user defined function needed to be created (see listing 3).

```
CREATE EXTERNAL TABLE users_urls_impressions (
    user_id STRING,
    url STRING,
    impressions INT
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';


INSERT OVERWRITE TABLE users_urls_impressions
SELECT user_id, domain, COUNT(1) AS impressions
FROM (
    FROM event_impressions e
    MAP e.page_url, e.user_id
    USING 'python /opt/etl/project/python/domain.py'
    AS domain, user_id
) domains
GROUP BY user_id, domain;
```

Listing 4. Hive SQL for retrieving list of domains per each user with the corresponding number of impressions.

Before proceeding to calculate u_{ij} matrix we need to calculate the total number of impressions in each of the domains.

```
CREATE EXTERNAL TABLE urls_impressions (
    url string,
    impressions int
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE urls_impressions
SELECT domain, COUNT(1) AS impressions
FROM (
    FROM event_impressions e
    MAP e.page_url
    USING 'python /opt/etl/project/python/domain.py'
    AS domain
) domains
GROUP BY domain;
```

Listing 5. Hive SQL query for calculating number of impressions in each of the domains

Finally we are ready to calculate TF-IDF weights per each user-domain pair.

```
CREATE EXTERNAL TABLE u_matrix_impressions (
    user_id string,
    url string,
    u_ij double
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE u_matrix_impressions
SELECT u.user_id, u.url, (LOG(u.impressions)+1) *
LOG(#domains/d.impressions)
FROM users_urls_impressions u
```

```
JOIN urls_impressions d ON d.url = u.url;
```

Where number of domains (#domains) can be calculated as:

```
select count(distinct url) from users_urls_impressions;
```

Selecting final set of TF-IDF values to external file:

```
hive -e 'SELECT * FROM u_matrix_impressions ORDER BY user_id' >                ⮒
./u_matrix_impressions.txt
```

Listing 6. Calculating TF-IDF values.

After calculating TF-IDF values and exporting them into local file we still need to perform additional transformation in order to shape that data into the final u_{ij} vector. These calculations are done with the use of the python script from the listing 7.

```
import sys
from itertools import groupby
import numpy
import optparse

import csv

def getReader(filePath):
    reader = csv.reader(open(filePath, "rb"), delimiter="¥t",
quoting=csv.QUOTE_NONE)
    return reader

def getUrlsAsSortedArray(fromFilePath):
    urlsDict = getFirstColumnData(fromFilePath)
    urls = sortValues(urlsDict)
    i = 0
    for urlKey in urls:
        urlsDict[urlKey] = i
        i += 1
    return urls, urlsDict

def getFirstColumnData(fromFilePath):
    urlsDict = {}
    reader = getReader(fromFilePath)
    for row in reader:
        url = row[1]
        urlsDict[url] = url
    return urlsDict

def sortValues(values):
    keys = values.keys()
    keys.sort()
    return map(values.get, keys)

def containsAnyValue(string, values):
    return True in [value in string for value in values]

def pivotData(inputFilePath, outputFilePath):
    csvFile = open(outputFilePath, 'w')
    csvWriter = csv.writer(csvFile, delimiter=" ",
        quotechar='', quoting=csv.QUOTE_NONE)

    urls, urlsDict = getUrlsAsSortedArray(inputFilePath)

    lastUserId = None
    vector = []
    userUrls = {}
    reader = getReader(inputFilePath)
    for row in reader:
        userId = row[0]
        if (userId == "NULL"):
            continue
        if lastUserId != userId and lastUserId != None:
            vector.append(lastUserId) # user ID
            vector.append(len(urls)) # size of the vector
            # create sparse vector
            for urlKey in userUrls:
                if not containsAnyValue(userUrls[urlKey], '¥/,='):   ⮒
                    vector.append(str(urlsDict[urlKey]) + ":"
                       + userUrls[urlKey])
            try:
                csvWriter.writerow(vector)
            except:
                print str(vector)
            vector = []
            userUrls = {}
        userUrls[row[1]] = row[2]
        lastUserId = userId

    csvFile.close()

if __name__ == '__main__':
    usage = "usage: %prog inputFile outputFile"
    parser = optparse.OptionParser(usage=usage)
    options, args = parser.parse_args()
    if len(args) == 2:
        inputFile = args[0]
        outputFile = args[1]
    else:
        print usage
        sys.exit(1)

    pivotData(inputFile, outputFile)
```

Listing 7. Python script (pivot_data.py)

Script from listing 7 creates a large file (many GB in size) which contains over one hundred thousand dimensions (one dimension for each of the domains) and over 10 millions of vectors.

6.3 Clustering user vectors

In the clustering phase we partition users' vectors into subsets where data in each subset is similar according to some defined metric. Yan et al. (2009) do not specify which metric was used in creating clusters. We assume here that the same metric which is used for calculating the similarity between and within-ads is used for clustering, that is the Cosine metric.

In order to perform clustering on the dataset of this size we use Apache Mahout – a data mining tool which is designed specifically to work in distributed environment across multiple CPUs or across clusters of machines. In this case Mahout performs all calculations running on Amazon EC2 cluster distributed across five large servers, each of them with 8 cores.

We divide clustering process into three parts – preparation of the data, clustering and analyzing the output. Data preparation is an important and necessary step. Mahout understands only specific format and therefore our data needs to be transformed into special vectors before we can proceed with the clustering. We transform data into Mahout sparse vectors to store only non-zero values and in this way save disk space and speed up calculations.

```
public class InputMapper extends Mapper<LongWritable, Text, Text,
VectorWritable> {

    private static final Pattern SPACE = Pattern.compile(" ");
    private static final Pattern COLON = Pattern.compile(":");

   private Constructor<?> constructor;



@Override
protected void map(LongWritable key, Text values, Context context) throws
IOException,   InterruptedException {

    String[] numbers = InputMapper.SPACE.split(values.toString());
    SequentialAccessSparseVector sparseVector = null;
    String keyName = "";
    int vectorSize = -1;
    for (String value : numbers) {
      if (keyName.equals("")) {
          keyName = value;
          continue;
      } else if (vectorSize == -1) {
          vectorSize = Integer.parseInt(value);
          sparseVector = new SequentialAccessSparseVector(vectorSize);
          continue;
      } else if (value.length() > 0) {
          String[] valuePair = InputMapper.COLON.split(value);
          if (!valuePair[1].equals("NULL")) {
            sparseVector.setQuick(Integer.parseInt(valuePair[0]),
Double.valueOf(valuePair[1]));
          }
      }
    }

    if (sparseVector != null) {
        try {
          Vector result = new NamedVector(sparseVector, keyName);
          VectorWritable vectorWritable = new VectorWritable(result);
          context.write(new Text(String.valueOf(vectorSize)),
vectorWritable);
        } catch (Exception e) {
          throw new IllegalStateException(e);
        }
    }
  }

}
```

Listing 8. Mahout code, written in Java, for transforming the sparse vector into Mahout sequence.

Once the data is correctly prepared for Mahout analysis we can execute the Mahout clustering mechanism.

```
$python /opt/etl/project/python/pivot_data.py ./u_matrix_impressions.txt
./pivoted_impressions.txt
hadoop fs -put ./pivoted_impressions.txt testdata

$ mahout org.apache.mahout.clustering.codes.kmeans.PrepareVector --input
testdata --output sequencedata

$ mahout org.apache.mahout.clustering.codes.kmeans.Job
--input sequencedata --distanceMeasure
org.apache.mahout.common.distance.CosineDistanceMeasure
--output output -k 160 -x 50 --clusters random-clusters
--clustering -overwrite

$ mahout org.apache.mahout.clustering.codes.kmeans.ClusterDumper
-p /user/hadoop/output/clusteredPoints
-s /user/hadoop/output/clusters-5 -o ./final-clusters.txt

hive -e 'LOAD DATA LOCAL INPATH "/mnt/project/mahout/final-clusters.txt
OVERWRITE INTO TABLE u_cluster'
```

Listing 9. Mahout command line parameters for executing clustering and dumping clustered data. We group all vectors into 160 clusters using 50 iterations.

At this stage we are ready to perform clustering on users vectors. Listing 9 shows all the commands which are required to prepare sparse vectors, execute clustering process, dump the resulting clustering and import final clusters back into Hive for further analysis. Note that due to the size limitations of this paper only the most important source code from modified files is included here. Specifically the ClusterDumper and the kmeans.Job classes needed to be modified in order to ensure that Mahout is able to perform analysis on our large dataset. Performing all analysis in the clustering step was the most computationally expensive part of the entire experiment as it took about 2 hours per each of the 50 clustering iterations.

6.4 CTR analysis on users clusters

In the previous step we clustered users using Apache Mahout project and then we imported those clusters into “u_cluster” table (see listing 10 for definition of “u_cluster”).

```
CREATE EXTERNAL TABLE u_cluster (
    user_id STRING,
    cluster_id INT
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';


CREATE EXTERNAL TABLE ads_nonopt_ctr (
    banner_id INT,
    ctr DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';


CREATE EXTERNAL TABLE ads_clusters_ctr (
    banner_id INT,
    cluster_id INT,
    ctr DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

CREATE EXTERNAL TABLE ads_opt_ctr (
    banner_id INT,
    ctr DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';
```

Listing 10. Hive definition of tables related to CTR calculations

We calculate the CTR of all ads, that is the CTR without any optimisation.

First we calculate “non-optimized” CTR per each ad

```
INSERT OVERWRITE TABLE ads_nonopt_ctr
SELECT i.banner_id, c.clicks / i.impressions AS ctr
FROM (
    SELECT COUNT(user_id) AS impressions, banner_id
    FROM event_impressions
    GROUP BY banner_id
) i LEFT OUTER JOIN
(
    SELECT COUNT (user_id) AS clicks, banner_id
    FROM event_clicks
    GROUP BY banner_id
) c ON (i.banner_id = c.banner_id);
```

Then we calculate CTR per each ad-cluster pair:

```
INSERT OVERWRITE TABLE ads_clusters_ctr
SELECT i.banner_id, i.cluster_id, c.clicks / i.impressions AS ctr
FROM (
    SELECT COUNT(i.user_id) AS impressions, i.banner_id, cl.cluster_id
cluster_id
    FROM event_impressions i JOIN u_cluster cl ON (i.user_id =
cl.user_id)
    GROUP BY i.banner_id, cl.cluster_id
) i LEFT OUTER JOIN
(
    SELECT COUNT(c.user_id) AS clicks, c.banner_id, cl.cluster_id
cluster_id
    FROM event_clicks c JOIN u_cluster cl ON (c.user_id = cl.user_id)
    GROUP BY c.banner_id, cl.cluster_id
) c ON (i.banner_id = c.banner_id and i.cluster_id = c.cluster_id)
WHERE i.impressions > 50;
```

Calculate CTR of optimized ads:
```
INSERT OVERWRITE TABLE ads_opt_ctr
SELECT banner_id, MAX(ctr) as max_ctr
FROM ads_clusters_ctr
GROUP BY banner_id;
```

Finally we calculate total CTR improvement:
```
SELECT SUM(d.delta_a) / COUNT(d.delta_a), COUNT(d.delta_a)
FROM (
    SELECT (o.ctr - no.ctr) / no.ctr AS delta_a
    FROM ads_nonopt_ctr no JOIN ads_opt_ctr o ON (no.banner_id =
o.banner_id)
) d;
```

Listing 11. CTR calculations performed by Hive

The code from listing 11 performs CTR analysis, calculates optimised and non-optimised CTR and the total CTR improvement.

6.5 Within- and between-ads user similarity

The within-ads and between-ads similarity is calculated only between users who ever clicked any ads. The goal of this analysis is to check if the users who clicked the same ads are move similar to each other than users who clicked different ads. We do not reuse clustering information in this part of experiment.

In order to calculate similarity between users we select only those users who clicked on any ad and calculate cosine similarity between them. Only around 0.02% of all users in our dataset clicked on ads and therefore the similarity analysis should be computationally less expensive and we can perform it in Python (see listing 12).

```
import sys
import math
import optparse
import csv

def getReader(filePath, delimiter=″ ″):
    reader = csv.reader(open(filePath, ″rb″), delimiter=delimiter,
quoting=csv.QUOTE_NONE)
    return reader

def getOneColumnData(fromFilePath, delimiter=″ ″, columnNumber = 0):
    dict = {}
    reader = getReader(fromFilePath, delimiter)
    for row in reader:
        record = row[columnNumber]
        dict[record] = record
    return dict

def dot(v1, v2):
    sum = 0.0
    for key in v1:
        # vectors are sparse
        if (v2.has_key(key)):
            sum += v1[key] * v2[key]
    return sum

def norm(v):
    sum = 0.0
    for key in v:
        sum += v[key] * v[key]
    return math.sqrt(sum)

def cosineSimilarity(v1, v2):
  return dot(v1, v2) / (norm(v1) * norm(v2))

def string2dict(vector):
    result = {}
    i = 0
    for value in vector:
        if i < 2: i += 1; continue
        if isinstance(value, str):
            s = value.split(':')
            result[int(s[0])] = float(s[1])
    return result

def filterClickUsers(inputFilePath, usersFile):
    tempFile = "/tmp/click_users_vectors.txt"
    csvFile = open(tempFile, 'w')
    csvWriter = csv.writer(csvFile, delimiter=" ",
        quotechar='', quoting=csv.QUOTE_NONE)

    usersDict = getOneColumnData(usersFile)
    reader = getReader(inputFilePath)
    for row in reader:
        userId = row[0]
        if userId in usersDict:
            csvWriter.writerow(row)
    csvFile.close()
    return tempFile

def runJob(inputFilePath, usersFile, outputFilePath):
    csvFile = open(outputFilePath, 'w')
    csvWriter = csv.writer(csvFile, delimiter="¥t",
        quotechar='', quoting=csv.QUOTE_NONE)

    usersDict = getOneColumnData(usersFile)

    # select only those users who we want to analyse
    filteredUsersPath = filterClickUsers(inputFilePath, usersFile)

    readerLeft = getReader(filteredUsersPath)
    for rowLeft in readerLeft:
        userIdLeft = rowLeft[0]
        if userIdLeft not in usersDict:
            continue
        # calculate similarity between this user and all next users in
the file
        readerRight = getReader(filteredUsersPath)
        for rowRight in readerRight:
            userIdRight = rowRight[0]
            if (userIdLeft == userIdRight) or (userIdRight not in     ⮒
usersDict):
                continue
            # remove user IDs before dot and norm vector calculations
            rowLeft[0] = rowRight[0] = 0.0
            sim = cosineSimilarity(string2dict(rowLeft),           ⮒
string2dict(rowRight))
            # save the output (save only non-zero results)
            if sim != 0.0:
                vector = [userIdLeft, userIdRight, sim]
                csvWriter.writerow(vector)

    csvFile.close()

if __name__ == '__main__':
    usage = "usage: %prog inputFile listOfUsers outputFile"
    parser = optparse.OptionParser(usage=usage)
    options, args = parser.parse_args()
    if len(args) == 3:
        inputFile = args[0]
        usersFile = args[1]
        outputFile = args[2]
    else:
        print usage
        sys.exit(1)

    runJob(inputFile, usersFile, outputFile)
```

Listing 12. Python script (similarity.py) for calculating cosine similarity between user vectors

In order to perform cosine similarity calculations we need to select all users who ever clicked on any ad. Listing 13 shows how to select all such users and how to call similarity.py script in order to retrieve the cosine similarity pairs.

Select list of users who clicked on any ad

```
$ hive -e 'SELECT DISTINCT user_id FROM event_clicks' > ./click_users.txt
```

Calculate similarities between users:

```
$ python /opt/etl/project/python/similarity.py ./pivoted_impressions.txt
./click_users.txt ./users_cossim.txt
```

Import user-pairs cosine similarities back into hive

```
CREATE EXTERNAL TABLE users_cossim (
    user_id1 STRING,
    user_id2 STRING,
    cossim DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

LOAD DATA LOCAL INPATH '/mnt/project/users_cossim.txt' OVERWRITE INTO
TABLE users_cossim;
```

Listing 13. Selecting "click users" and calculating cosine similarity with the use of similarity.py script

After calculating cosine similarity between all pairs of users we can calculate S_w and S_b.

```
CREATE EXTERNAL TABLE users_cossim_within (
    banner_id INT,
    sw DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE users_cossim_within
SELECT banner_id, 2 / count(uc.cossim) * (count(uc.cossim) - 1) *
sum(cossim) AS sw
FROM (
     SELECT uc.user_id, uc.banner_id, avg(uc.cossim) AS cossim
        FROM (
            SELECT ec.user_id, ec.banner_id, cossim
            FROM event_clicks ec JOIN users_cossim uc1 ON (ec.user_id =
uc1.user_id1)
            UNION ALL
            SELECT ec.user_id, ec.banner_id, cossim
            FROM event_clicks ec JOIN users_cossim uc2 ON (ec.user_id =
uc2.user_id2)
        ) uc
        GROUP BY uc.banner_id, uc.user_id) uc
GROUP BY banner_id;
```

Calculate total S_w:

```
SELECT SUM(sw) / COUNT(1) AS total_sw FROM users_cossim_within;
```

Listing 14. Hive S_w calculations

Although it is possible to use subselects in Hive it is often necessary to use many additional helper tables for storing intermediate results. Listing 15 shows such additional tables which are aiding us in calculating total S_b.

```
CREATE EXTERNAL TABLE banner_users (
    banner_id INT,
    user_id STRING
```

```
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE banner_users
SELECT banner_id, user_id
FROM event_clicks
GROUP BY banner_id, user_id;


CREATE EXTERNAL TABLE banner_pair_users (
    banner_id1 INT,
    banner_id2 INT,
    user_id1 STRING,
    user_id2 STRING
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE banner_pair_users
SELECT bu1.banner_id, bu2.banner_id, bu1.user_id, bu2.user_id
FROM banner_users bu1 JOIN banner_users bu2;

CREATE EXTERNAL TABLE banner_pair_users_cossim (
    banner_id1 INT,
    banner_id2 INT,
    user_id1 STRING,
    user_id2 STRING,
    cossim DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE banner_pair_users_cossim
SELECT bpu.banner_id1, bpu.banner_id2, bpu.user_id1, bpu.user_id2,
uc.cossim
```

```
FROM banner_pair_users bpu JOIN users_cossim uc ON (bpu.user_id1 =
uc.user_id1 AND bpu.user_id2 = uc.user_id2);

CREATE EXTERNAL TABLE banner_distinct_users_count (
    banner_id INT,
    number_of_users INT
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE banner_distinct_users_count
SELECT banner_id, COUNT(distinct user_id)
FROM event_clicks
GROUP BY banner_id;

CREATE EXTERNAL TABLE banner_cossim_between_sum (
    banner_id1 INT,
    banner_id2 INT,
    cossim_sum DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE banner_cossim_between_sum
SELECT bpuc.banner_id1, bpuc.banner_id2, SUM(bpuc.cossim)
FROM banner_pair_users_cossim bpuc
GROUP BY bpuc.banner_id1, bpuc.banner_id2;
```

Calculate S_b for each banner pairs:

```
CREATE EXTERNAL TABLE banner_cossim_between (
    banner_id1 INT,
    banner_id2 INT,
    cossim DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE banner_cossim_between
SELECT banner_id1, banner_id2, cossim_sum / (bduc1.number_of_users *
bduc2.number_of_users) AS bs
FROM banner_cossim_between_sum bcbs
JOIN banner_distinct_users_count bduc1 ON (bcbs.banner_id1 =
bduc1.banner_id)
JOIN banner_distinct_users_count bduc2 ON (bcbs.banner_id2 =
bduc2.banner_id);
```

Number of distinct banners (#db):

```
SELECT COUNT(distinct banner_id) AS db from event_clicks;
```

and finally we calculate total S_b:

```
SELECT SUM(cossim)/(#db * #db) from banner_cossim_between;
```

Listing 15. Hive SQL calculations for S_b

Note that S_w and S_b are stored in separate tables which makes it easy to retrieve those values later in order to perfom t-test between those values (Cox & Hinkley 1974). After calculating S_w and S_b we can calculate the R measure, see listing 16.

```
CREATE EXTERNAL TABLE banner_r (
    banner_id1 INT,
    banner_id2 INT,
    r DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';
```

We add 0.001 in order to avoid dividing by zero

```
INSERT OVERWRITE TABLE banner_r
SELECT sb.banner_id1, sb.banner_id2, (sum(sw1.sw) + SUM(sw2.sw)) + 0.001
/ (2 * SUM(sb.cossim) + 0.001) AS r
FROM banner_cossim_between sb
JOIN users_cossim_within sw1 ON (sw1.banner_id = sb.banner_id1)
JOIN users_cossim_within sw2 ON (sw2.banner_id = sb.banner_id2)
GROUP BY sb.banner_id1, sb.banner_id2;

SELECT MIN(sw) FROM users_cossim_within WHERE sw != 0;

SELECT COUNT(DISTINCT banner_id) FROM event_clicks;
```

Finally we calculate total R:

```
select count(distinct banner_id1) as db FROM banner_r;
select sum(r)/(#db * #db) from banner_r;
```

Listing 16. Hive R calculations

6.6 F-measure, Precision and Recall

In order to calculate F-measure we need first to calculate precision and recall. All these measure will help us to evaluate how effective was the improvement in CTR. First we calculate the recall, see listing 17.

```
CREATE EXTERNAL TABLE ad_all_clusters_rec (
    ad_id INT,
    cluster_id INT,
    rec DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE ad_all_clusters_rec
SELECT cc.banner_id, cc.cluster_id, cc.clicks / cl.clicks AS rec
```

```
FROM
(
    SELECT COUNT(c.user_id) AS clicks, c.banner_id, cl.cluster_id
cluster_id
    FROM event_clicks c JOIN u_cluster cl ON (c.user_id = cl.user_id)
    GROUP BY c.banner_id, cl.cluster_id
) cc JOIN
(
    SELECT COUNT(user_id) AS clicks, banner_id
    FROM event_clicks
    GROUP BY banner_id
) cl
ON (cc.banner_id = cl.banner_id);


We get the optimized clusters (g*)
CREATE EXTERNAL TABLE ads_clusters_rec (
    ad_id INT,
    cluster_id INT,
    rec DOUBLE
 )
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE ads_clusters_rec
SELECT ac.ad_id, acl.cluster_id, ac.rec
FROM ad_all_clusters_rec ac
JOIN ads_opt_clusters acl ON (ac.ad_id = acl.banner_id and ac.cluster_id
= acl.cluster_id);

Calculate total average Recall:
SELECT SUM(rec) / COUNT(1) FROM ads_clusters_rec;
```

Listing 17. Hive Recall calculations

After recall we calculate precision, that is a measure to present only relevant items. In our case it is simply equal the CTR from each clusters.

Create table which contains ads clusters with maximum CTR:

```
CREATE EXTERNAL TABLE ads_opt_clusters (
    banner_id INT,
    cluster_id DOUB:E
 )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE ads_opt_clusters
SELECT acc.banner_id, acc.cluster_id
FROM ads_clusters_ctr acc join ads_opt_ctr aoc ON (acc.banner_id =
aoc.banner_id)
WHERE acc.ctr = aoc.ctr;

First calculate precision just for optimised clusters:
CREATE EXTERNAL TABLE ads_clusters_prec (
    banner_id INT,
    cluster_id INT,
    prec DOUBLE
 )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE ads_clusters_prec
SELECT ac.banner_id, acl.cluster_id, ac.ctr AS prec
FROM ads_opt_ctr ac
JOIN ads_opt_clusters acl ON (ac.banner_id = acl.banner_id);

Calculate total average Prec:
SELECT SUM(prec) / COUNT(1) FROM ads_clusters_prec;
```

Listing 18. Hive Precision calculations

At this stage we are ready to calculate the F-measure which combines both precision and recall in the harmonic mean.

```
CREATE EXTERNAL TABLE ads_clusters_f (
    ad_id INT,
    cluster_id INT,
    f DOUBLE
 )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '¥t';

INSERT OVERWRITE TABLE ads_clusters_f
SELECT acp.banner_id, acp.cluster_id, (2 * acp.prec * acr.rec) /
(acp.prec + acr.rec) AS f
FROM ads_clusters_prec acp
JOIN ads_clusters_rec acr ON (acr.ad_id = acp.banner_id AND
acr.cluster_id = acp.cluster_id);
```

Calculate total F measure:

```
select sum(f) / count(1) from ads_clusters_f;
```

Listing 19. Hive F-measure calculations

7. Results

The results were calculated for 160 clusters of users. The research done by Yan et al. (2009) calculated results for 4 groups of clusters of 20, 40, 80 and 160 clusters each and achieved best CTR improvement for 160 clusters. Due to limited computational resources and much larger data set this experiment was able to calculate the results only for 160 clusters.

7.1 Within- and between-ads user similarity

Similarities S_w and S_b were calculated in order to assess if users who clicked the same ads (within-ads similarity) are indeed more similar to each other than users who clicked different ads (between-ads similarity). Table 2 present results of calculating S_w and S_b.

	S_w	S_b	R
80 millions records	**0.259**	**0.041**	**1.4**

Table 2. Within- and between-ads user similarity.

From the results in table 2 we can see that total averaged S_w is much higher than total S_b. An independent t-test confirmed the difference to be highly significant $t = 2.73 \times 10^{-273}, p < 0.001$.

The results confirmed our initial hypothesis that users who clicked the same ads are more similar to each other than users who clicked different ads. However, S_b in our case is almost twice as big as the same results from the research by Yan et al. (2009). That suggests that the users who clicked different ads are still quite similar to each other. This difference may be caused by the fact that in this experiment users are clustered based on the domains they visited and not based on the URLs. Therefore users who visited different URLs are often within the same domain and therefore are more likely to be more similar to each other. High S_b may explain rather low R measure which is an order of magnitude lower than the results achieved by Yan et al. (2009).

As mentioned previously the R measure calculation uses additional parameter in order to ensure that all R values are possible to calculate. The resulting R does not change much (variations within 0.1%) when d is changed from 0.001 to 0.0001 so we can assume that this parameter is safe to use and it doesn't change the final results significantly.

7.2 CTR improvement

From the perspective of the ad network one of the most important evaluation metric is the CTR improvement. Clustering users into 160 clusters improved overall CTR by 909%. This number is approximately within the range of improvement achieved by Yan et al. (2009), however it is rather high. One of the possible explanations is that search advertising CTR is typically much higher than display advertising CTR and therefore it may be easier to improve display advertising lower CTR.

	CTR improvement
K-means (160 segments)	**909%**

Table 3. Total CTR improvement

It is important to discuss how exactly this improvement was calculated. The total averaged CTR averages all best CTR achieved for every ad in all clusters and therefore we do not take into account how much of the ads were delivered into such a user segment. It is possible that some segments may capture only few percents of all impressions and clicks of a given ad and still can yield the highest CTR. In order to minimise such a possibility our calculations ensured that each ad-cluster has at least 50 impressions before it is included in the final CTR improvement measure.

It would be an interesting project to further explore the possibilities of improving the total CTR with the restriction that each ad needs to deliver specific number of impressions.

7.3 Precision, Recall and F-measure

Table 4 presents information on recorded precision, recall and F-measure on 160 clusters.

K-means (160 segments)	**Precision**	**83.4%**
	Recall	**8.07%**
	F-measure	**0.12**

Table 4. Precision, recall and F-measure.

K-means on 160 clusters achieved much higher precision than Yan et al. (2009) but also much lower recall. Small recall indicates that only about 8% of all impressions were captured in the clusters with highest CTR. In other words even though the improvement in CTR was large it contained only 8% of all clicks recorded in the dataset. It should be possible to decrease the CTR on the cost of recall and yield lower CTR improvements but applied to a higher number of clicks.

8. Discussion

This study considered the impact of behavioural targeting techniques on online display advertising. More specifically, we investigated whether simulating delivery of traffic to chosen clusters will increase the overall CTR of all ads. We examined the data using different evaluation metrics such as: user similarity, precision, recall, F-measure and we used the t-test to confirm the significance of the results. The experimental design was implemented with the help of scalable data mining libraries which allowed successful analysis of large body of data.

This study was motivated by Yan et al. (2009) which found that behavioural targeting in search advertising can yield up to 670% increase in the overall CTR. We performed the systematic study of clickstream logs of a commercial ad network in which we found that the overall CTR can be increased as much as 909%.

This research shows that it is possible to achieve higher CTR by the means of clustering users. However there are a few issues with focusing only on the overall CTR. In calculating CTR improvement we ignore how much impressions were delivered to each cluster. In the display advertising model advertisers typically pay per number of impressions and not per clicks. Therefore, to effectively use these techniques in real-life we should take into account not only CTR but the ad network contracts as well, that is a number of ad views which ad network agreed to deliver.

Yet another issue is that the data was clustered and tested on the same day. This shows theoretical improvements given that we would know to which clusters users should be delivered to. However, in reality the users would be clustered on the data from the previous day and ads would be targeted to such clusters the day after. The same applies to choosing which clusters yield highest CTR. It is relatively easy to check the CTR for each ad-cluster pairs but predicting beforehand which clusters are the best candidates would require additional work.

The results of our current study suggest that the behavioural targeting has an enormous potential to improve the effectiveness of online display advertising. While the clustering method used in this experiment was limited to a post-hoc analysis of data collected within a single day, it is highly probable that a similar model would perform very well in reality. We base this conclusion on previous research that showed similar improvements following BT applied to search advertising (Yan et al. 2009). Furthermore, our model was rather simple; it should be possible to achieve even better results using more advanced user segmentation methods.

9. References

Brettel, M., & Spilker-Attig, A. (2010). Online advertising effectiveness: a cross-cultural comparison. *Journal of Research in Interactive Marketing* 4(3), 176-196.

Chen, Y., Pavlov, D., & Canny, J.F. (2009). Large-scale behavioral targeting. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 209-218.

Cox, D.R., & Hinkley, D.V. (1974). *Theoretical statistics*. Chapman and Hall, London.

Dreze, X., & Hussherr, F.X. (2003). Internet advertising: is anybody watching? *Journal of Interactive Marketing*, 17(4), 8-23.

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. *In Proceedings of the Sixth Symposium on Operating System Design and Implementation,* 137–150.

Hripcsak, G., & Rothschild, A.S. (2005). Agreement, the F-Measure, and reliability. *Information Retrieval Journal of the American Medical Informatics Association*, 2, 296-298.

Joachims, T., Granka, L., Pan, B., Hembrooke, H. & Gay, G. (2005) Accurately interpreting clickthrough data as implicit feedback. *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, 154-161.

Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R., & Wu. A. (2000). An efficient K-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24, 881-892.

Karypis., G. CLUTO: a software package for clustering high-dimensional data sets. University of Minnesota, Department of Computer Science.

Manchanda, P., Dubé, J., Goh., K.Y., & Chintagunta, P.K. (2006). The Effect of Banner Advertising on Internet Purchasing. *Marketing Research*, 43(1).

Papineni, K. (2001). Why inverse document frequency? In NAACL '01: *Second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies 2001*, 1-8.

Pincas, S. & Loiseau, M. (2008). *A History of Advertising*, Taschen, Los Angeles.

Ratnaparkhi, A. (2010). Finding predictive search queries for behavioral targeting. *In ADKDD'10, The 4th International Workshop on Data Mining and Audience Intelligence for Advertising*.

Rijsbergen, C.J. (1979). *Information Retrieval*. London: Butterworths, 2nd Edition.

Salton, G., & Buckley., C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management: an International Journal*, 24, 513-523.

Wolin, L.D. (2003). Gender Issues in Advertising An Oversight Synthesis of Research: 1970 2002. *Journal of Advertising Research*, 43(1), 111-129.

Yan , J., & Liu, N., & Wang, G., & Zhang, W., & Jiang, Y., & Chen, Z. (2009). How much can Behavioural Targeting Help Online Advertising? *Proceedings of the 18th international conference on World Wide Web. Madrid*, Spain.