B.E (Information Technology) Final Year Project Report on

ADVERTISEMENT BLOCK USING MACHINE LEARNING

By

ADITI DEO (001511001045)
&
SHUBHAM DUTTA (001511001010)

Under the guidance of

Mr Utpal Kumar Ray

Department of Information Technology

Faculty of Engineering and Technology

Jadavpur University
Kolkata, India
2018 – 2019

Declaration by the Students

We, Aditi Deo and Shubham Dutta, hereby declare that this project report entitled "Advertisement using Machine Learning" contains only the work completed by us as a part of the Bachelor of Engineering course, during the year 2018 – 2019, under the supervision of Mr Utpal Kumar Ray, Department of Information Technology, Jadavpur University.

All information, materials and methods that are not original to this work have been properly referenced and cited. All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

We also declare that no part of this project work has been submitted for the award of any other degree prior to this date.

Signature:	Signature:
Date:	Date:

Certificate

This is to certify that the project entitled “Advertisement Block Using Machine Learning” was carried out by Aditi Deo(001511001045) and Shubham dutta(001511001010) and submitted to the Jadavpur University during the year 2018-19 for the award of the degree of Bachelors of Engineering (Information Technology), is a bona fide record of work done by them under my supervision.

Signature of the supervisor:

Date :

Acknowledgement

We would like to thank our project guide Mr Utpal Kumar Ray, Assistant Professor of Jadavpur University for their guidance and support and the faculty members of the department for their cooperation. We extend our thanks to our classmates who have helped us in our hours of need.

Abstract

Advertisements simultaneously provide both economic support for most free web content and one of the largest annoyances to end users. Furthermore, the modern advertisement ecosystem is rife with tracking methods which violate user privacy.

A natural reaction is for users to install ad blockers which prevent advertisers from tracking users or displaying ads. Traditional ad blocking software relies upon hand-crafted filter expressions to generate large, unwieldy regular expressions matched against resources being included within web pages. This process requires a large amount of human overhead and is susceptible to inferior filter generation.

We propose an alternate approach which leverages machine learning to bootstrap a superior classifier for ad blocking with less human intervention. We show that our classifier can simultaneously maintain an accuracy similar to the hand-crafted filters while also blocking new ads which would otherwise necessitate further human intervention in the form of additional handmade filter rules.

TABLE OF CONTENTS

1.0 INTRODUCTION

For almost as long as the commercial world wide web has existed, annoying advertisements have competed with desired content for users' attention. To combat these annoyances , users have created methods for filtering out the unwanted ads

While advertisements are necessary to support the business model of most content providers, they have long been considered extremely annoying and intrusive to most users . Besides just serving ad content, ad networks use the advertisements to retrieve information about the user and their browsing habits. Additionally, many ads are being used as “malvertisements" that serve malware to unsuspecting victims.

There are various browser extensions that block advertisement resources and hide related HTML content. Currently, AdBlock Plus , AdBlock , Disconnect, Ghostery and some proprietary software such as AdMuncher are the most well-known and popular with effective results. Modern ad blocker implementations have several technical shortcomings.

Currently deployed solutions all make use of regular expressions written by contributing users to target specific ads or ad placement schemes. These regular expressions are then compiled into a list that is injected into the browser. Individual resources are matched against the list to test for inclusion in the page. There is substantial human overhead as no automated process exists to generate these unwieldy blacklists.

This project presents an alternative technique for detecting and blocking advertisement resources. We find that using the machine learning approach to train a classifier that detects advertisements based on historical regular expression lists is the most successful, and show that this classifier is able to both classify current ads with high accuracy as well as detect new ads which might normally necessitate human intervention in the form of additional hand written filters.

2.0 PRE REQUISITES

(Overview of required pre requisites are explained below.)

- Python 2.7 with pip to be used in windows.(libraries are installed using "pip install x" command). [11]
- Mysql should be installed.
- Flask should be configured.

Some pre requisites concept are explained henceforth.

2.1 FLASK

Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

What is Flask?

Flask is a web application framework written in Python.

In order to test **Flask** installation, type the following code in the editor as **Hello.py**

```
from flask import Flask
app = Flask(__name__)

@app.route('/')

def hello_world():
  return 'Hello World''

if __name__ == '__main__':
  app.run()
```

Importing flask module in the project is mandatory. An object of Flask class is our **WSGI** application.

Flask constructor takes the name of **current module (__name__)** as argument.

The **route()** function of the Flask class is a decorator, which tells the application which URL should call the associated function.

```
app.route(rule, options)
```

- The **rule** parameter represents URL binding with the function.
- The **options** is a list of parameters to be forwarded to the underlying Rule object.

In the above example, **'/'** URL is bound with **hello_world()** function. Hence, when the home page of web server is opened in browser, the output of this function will be rendered.

Finally the **run()** method of Flask class runs the application on the local development server.

```
app.run(host, port, debug, options)
```

All parameters are optional

Sr.No.	Parameters & Description
1	**host** Hostname to listen on. Defaults to 127.0.0.1 (localhost). Set to "0.0.0.0" to have server available externally
2	**port** Defaults to 5000
3	**debug** Defaults to false. If set to true, provides a debug information
4	**options** To be forwarded to underlying Werkzeug server.

The above given **Python** script is executed from Python shell.

```
Python Hello.py
```

A message in Python shell informs you that

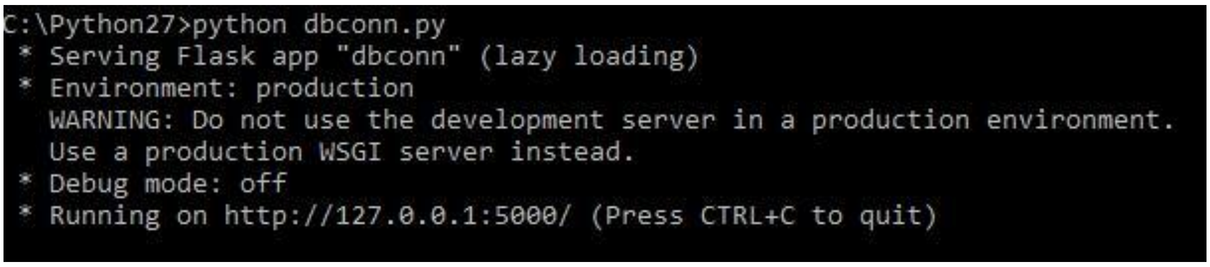
```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Open the above URL **(localhost:5000)** in the browser. **‘Hello World’** message will be displayed on it.

Externally Visible Server

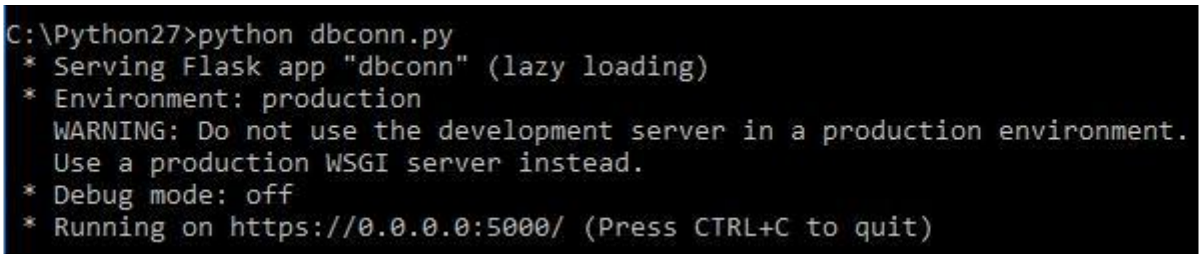
If you run the server you will notice that the server is only available from your own computer, not from any other in the network. This is the default because in debugging mode a user of the application can execute arbitrary Python code on your computer. If you have debug disabled or trust the users on your network, you can make the server publicly available.

Just change the call of the run() method to look like this:
app.run(host='0.0.0.0')
This tells your operating system to listen on a public IP.



```
C:\Python27>python dbconn.py
 * Serving Flask app "dbconn" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Running on server not externally visible. {**app.run()**}



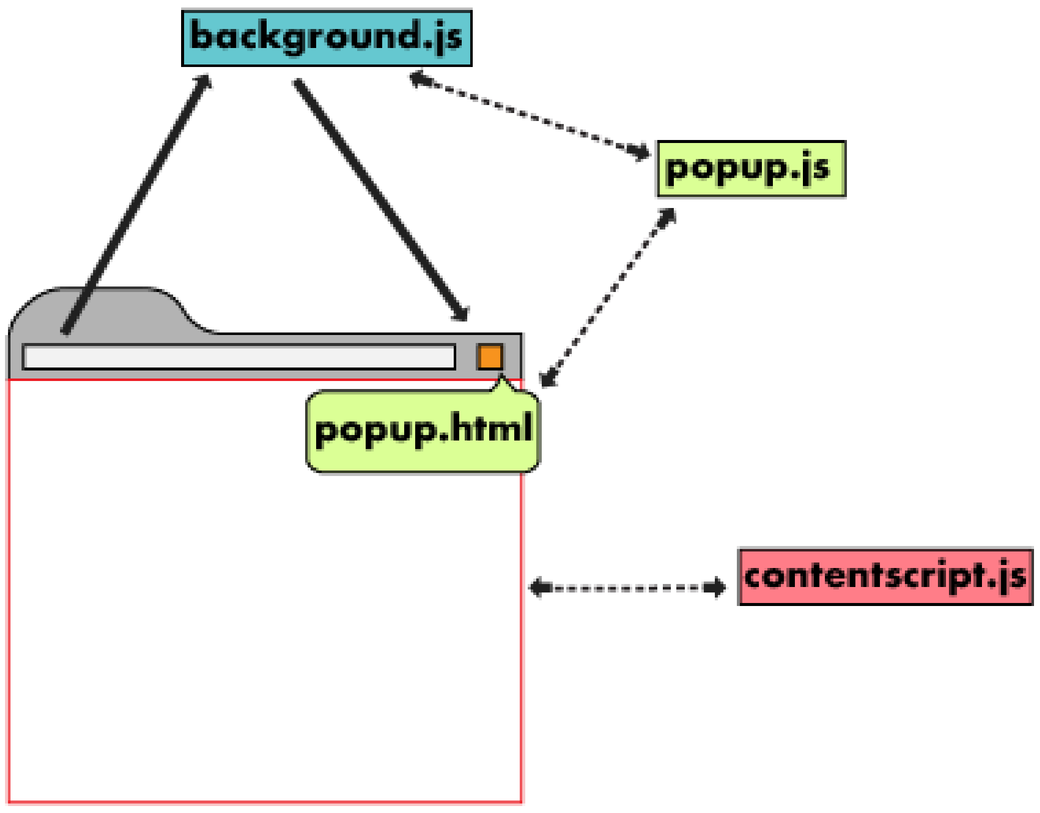
```
C:\Python27>python dbconn.py
 * Serving Flask app "dbconn" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on https://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Running on server not externally visible. { **app.run(host='0.0.0.0',ssl_context='adhoc')}**

2.2 EXTENSIONS

Chrome extension is an integral part of this project.

What are extensions ?
• Extensions are small software programs that customize the browsing experience.
• They enable users to tailor Chrome functionality and behaviour to individual needs or preferences.
• They enable user to manipulate DOM of web page.
• An extension must fulfil a single purpose that is narrowly defined and easy to understand. A single extension can include multiple components and a range of functionality, as long as everything contributes towards a common purpose.



3.0 PROBLEM DESCRIPTION

Advertisements are necessary to support the business model of most content providers, they have long been considered extremely annoying and intrusive to most users. Besides just serving ad content, ad networks use the advertisements to retrieve information about the user and their browsing habits. Additionally, many ads are being used as malvertisements that serve malware to unsuspecting victims . Hence, advertisement blocking is definitely a non-trivial defence against the security risks present on the internet of today. There are various browser extensions that block advertisement resources and hide related HTML content. Currently, AdBlock Plus , AdBlock , Disconnect , Ghostery and some proprietary software such as AdMuncher are the most well-known and popular with effective results.

Modern ad blocker implementations have several technical shortcomings. Currently deployed solutions all make use of regular expressions written by contributing users to target specific ads or ad placement schemes. These regular expressions are then compiled into a list that is injected into the browser. Individual resources are matched against the list to test for inclusion in the page. In AdBlockPlus's EasyList , the number of filters is on the order of tens of thousands. The filter list can be extremely specific and thus a page element or URL that is clearly ad-related may not be classified by the list because a certain substring is not found. In fact, most blocking is done by keeping an additional blacklist of third-party advertisers. There is substantial human overhead as no automated process exists to generate these unwieldy blacklists.

The objective is to present an alternative technique for detecting and blocking advertisement resources

4.0 APPROACH

Our approach to classifying ad URLs relies heavily on the AdBlockPlus classification system. AdBlockPlus is widely regarded as the premier ad blocking extension with EasyList being its most common advertisement filter list. EasyList has well over 12 million subscribers today and is often updated on a weekly basis . EasyList can filter unwanted content via URL filters, DOM element filters, and third-party advertisement domain filters. Our goals :

- To overcome the problem of having to manually update the EasyList filter list.
- To achieve accuracy competitive with current manual filtering while simultaneously detecting ads which would be missed by contemporary manual filters.

We use a supervised learning approach to bootstrap the process of learning whether a URL is ad-related (by using EasyList to label data) and subsequently block ad-based url resource from its corresponding web page.

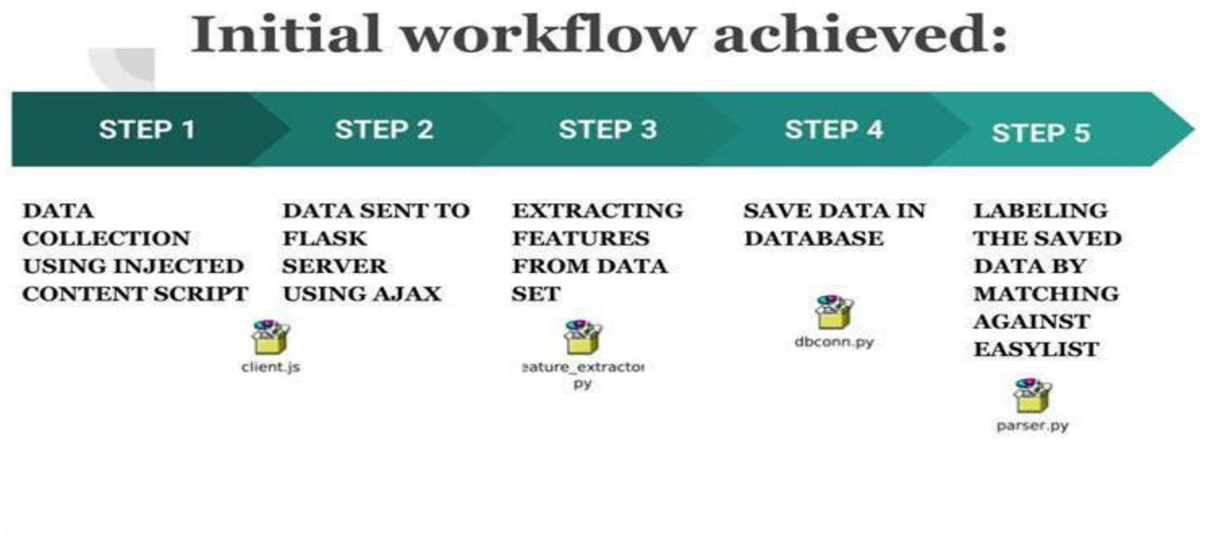
Our classification problem is a binary classification problem on URLs where positive examples are ad-related and negative examples are non-ad-related.

Positive examples are resources requested to serve ads - typically requested from within an ad element - and negative examples are any other resource requested on a page that is not related to serving ads.

We use 7 features for each URL, the majority of which are binary along with a few which are real-valued. Our feature space consists of features derived from the anatomy of the URL itself as well as some of the originating page properties from which the URL was requested.

Workflow of complete model is explained below.

4.1 WORKFLOW



Initial workflow achieved:

STEP 1

DATA COLLECTION USING INJECTED CONTENT SCRIPT

client.js

STEP 2

DATA SENT TO FLASK SERVER USING AJAX

STEP 3

EXTRACTING FEATURES FROM DATA SET

eature_extractor py

STEP 4

SAVE DATA IN DATABASE

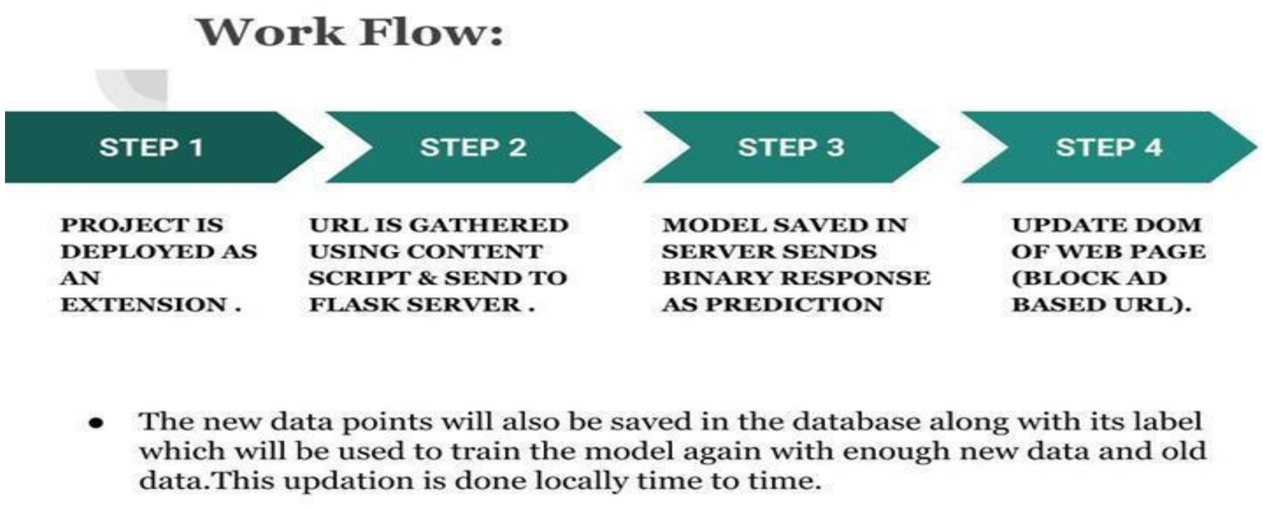
dbconn.py

STEP 5

LABELING THE SAVED DATA BY MATCHING AGAINST EASYLIST

parser.py

- Locally train the model using trainer.py and save the model in server



Work Flow:

STEP 1

PROJECT IS DEPLOYED AS AN EXTENSION .

STEP 2

URL IS GATHERED USING CONTENT SCRIPT & SEND TO FLASK SERVER .

STEP 3

MODEL SAVED IN SERVER SENDS BINARY RESPONSE AS PREDICTION

STEP 4

UPDATE DOM OF WEB PAGE (BLOCK AD BASED URL).

- The new data points will also be saved in the database along with its label which will be used to train the model again with enough new data and old data.This updation is done locally time to time.

INITIAL WORKFLOW ACHIEVED:

- Collect data by injecting content script into the web page through extension.[Refer APPENDIX A(find() function)]:As content script can manipulate the DOM of the web page the browser goes, hence the code to search and store all url tags is written in content script which gets injected as the DOM gets loaded.[10]

- Send data to flask server using AJAX.[Refer appendix A (req() function)]:Data point is saved as an JSON object which needs to be converted to string to send it to flask server.[8]

  {In client.js(client) req() function will make contact to "store()" module of dbconn.py(flask server)}

- Extract features from the dataset.[Refer APPENDIX C]:Features are extracted by parsing various components of url and also considering url's context of web page.{feature_extractor.py is imported in dbconn.py}

- Save data in database.[Refer APPENDIX B]: Before moving further, Please make sure you have the following in place: –

  - **Username** and **password** that you need to connect MySQL
  - **MySQL database table name** in which you want to insert data

  To perform a SQL INSERT query from Python, you just need to follow these simple steps: [7]

  - Install MySQL Connector Python using pip.
  - First, Establish a MySQL database connection in Python.
  - Then, Define the SQL INSERT Query (here you need to know the table's column details).
  - Execute the INSERT query using the **cursor.execute()** and get a number of rows affected.
  - After successful execution of a query, Don't forget to commit your changes to the database.
  - Close the MySQL database connection.
  - Most important, Catch SQL exceptions if any.
  - At last, verify the result by selecting data from MySQL table.

- Label the saved data.[refer appendix D]:-We label the data by matching it with the filter list used by adblock plus(Easylist).

FINAL WORKFLOW

- First we train the model on our data using various classification algorithms .We pick the best classifier model and save it in the flask server using pickle keyword[6]
  This happens locally.[Refer APPENDIX E]

- Then we deploy the whole model as chrome extension.[Refer APPENDIX F]

- Data points are gathered using content script and again sent to flask server(using AJAX) where our model is saved.
  {In client.js(client) pred() function will make contact to "predict()" module of dbconn.py(flask server)} .[Refer APPENDIX A(pred() function)]]

- Data points reaching to the server is parsed and features are extracted (pre processing is done) to generate a feature vector. [Refer APPENDIX C]

- The model saved in the server will get this feature vectior as input and it will send the response back to the content script. [Refer APPENDIX C("predict()" module)]

- If the response is positive i.e. data point is ad-based then we will block that html element. .[Refer APPENDIX A(pred() function)]

Hence the model is deployed as an extension. So a user just needs to install the extension in his/her browser. And the extension will perform the job of data extraction and it will contact with the IP address of the machine where flask server is running.

[Pre-requisite:-Externally Visible Server

If you run the server you will notice that the server is only available from your own computer, not from any other in the network. This is the default because in debugging mode a user of the application can execute arbitrary Python code on your computer. If you have debug disabled or trust the users on your network, you can make the server publicly available.

Just change the call of the run() method to look like this:

```
app.run(host='0.0.0.0')
```

This tells your operating system to listen on a public IP.]

Hence all the external users need to communicate with the IP of the host machine where server is running.

Thus all users just need to write that IP address in their client.js in their pred() and req() function when they are making connection using AJAX.

SAMPLE AJAX:-

```
xhttp.open("POST", "http://localhost:5000/store", true);
xhttp.setRequestHeader("Content-type", "application/json/text");
xhttp.send(data);
```

Instead of localhost we need to write that IP address.

5.0 DATA COLLECTION

To generate a complete dataset comprising of training and test data :-

- We visited top websites to extract maximum url resources using chrome extension.
- The corresponding labelling is done by matching the extracted urls with Easylist(filter list used by adblock plus) in order to identify whether an url resource is ad-based or not.[Refer Appendix D].

For extraction of URL resources from a webpage we need to access the DOM of webpage . Hence we have used chrome extension.

Extensions are made of different, but cohesive, components. Components can include background scripts, content scripts, an options page, UI elementsand various logic files. Extension components are created with web development technologies: HTML, CSS, and JavaScript. An extension's components will depend on its functionality and may not require every option.

In our case , as soon as the webpage is loaded content script is injected which is used to extract all url resources from that webpage.(We can also invoke the content script by onclick event on icon of extension, but for that we need to use background script which will pass message to invoke content script).

Now all the extracted url data point is send to the flask server where further processing is done . Data points are send using AJAX.[refer Appendix A]

6.0 FEATURE EXTRACTION

For the classifier to be effective, we attempted to identify features that may differentiate an ad-related URL from a non ad-related URL .
We perform feature extraction when the data point is sent to server and then subsequently tuple comprising of features, label class and url itself is saved to mysql database.
Some features are extracted while extracting urls were as some are collected after complete extraction of url(when it‟s sent to server).[Refer Appendix C]

Below is a summary of the key feature categories we used to train our classifier:

- Ad-related keywords: A URL is more likely to be ad-related than not if it contains certain keywords. We check for: `ad', `advert', `popup', `banner', `sponsor', `iframe', `googlead', `adsys', and `adser'.

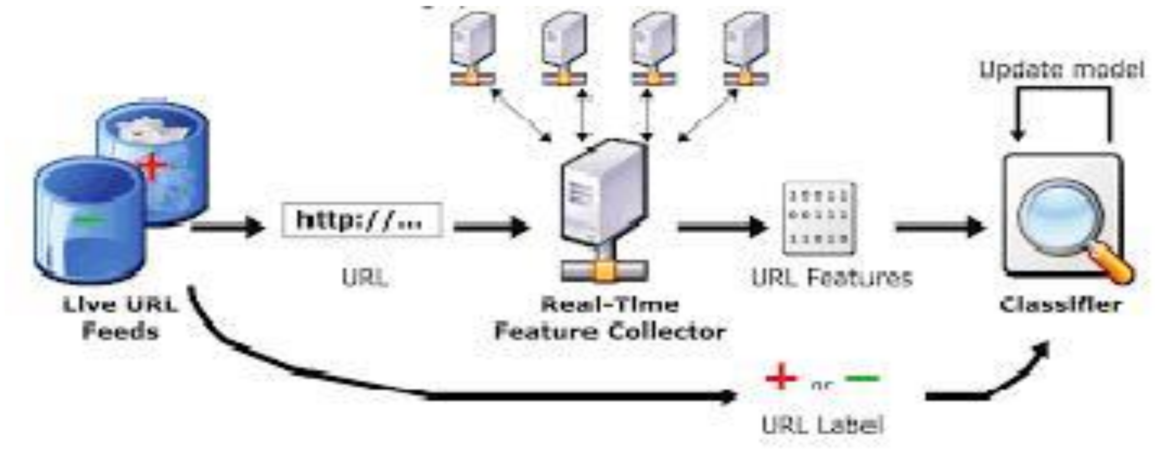
- Related to the original page: The second feature captures whether the requested URL is on the same domain or subdomain of the original page as URLs are less likely to be ad-related if they are requested on the same domain as the original page. This feature is not error-proof however, as it can backfire in certain situations like YouTube or Google ads where sometimes the ad is hosted on the base domain itself.

- In an iframe container.:This feature is binary and indicates whether the incriminating URL was requested from within an iframe either in the context of the page or in the context of nested iframes

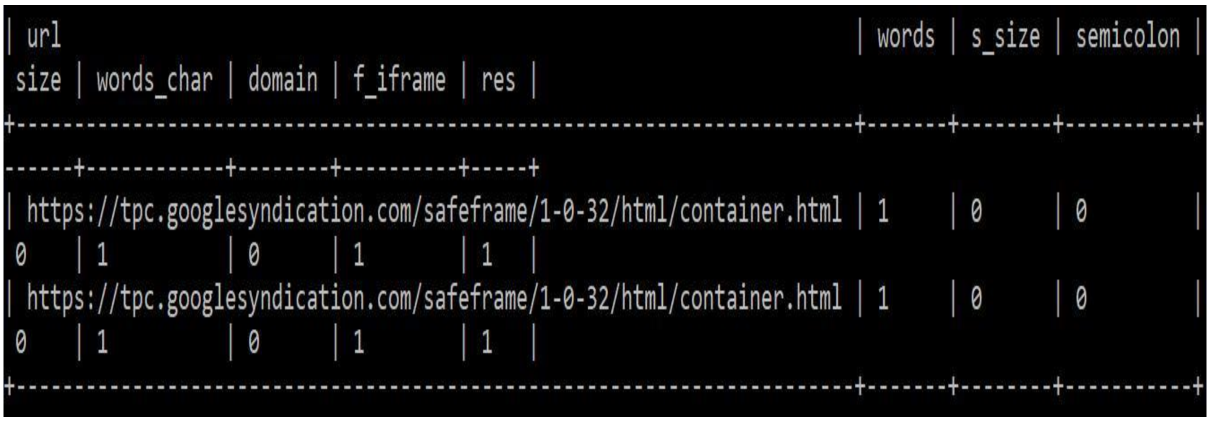
- Lexical features: This feature set contains two features that come from the character arrangements in the URL itself. First, whether it contains semicolons to separate parameters. It could either be in the place of actual query parameters usually separated by `&' or a string of semicolon parameters as part of the query parameters or even the path itself.
  The second feature in this set is whether the URL contains valid query parameters. Valid query parameters are formatted by being placed after the `?' and then each parameter is separated by a `&'. Many URLs that were identified as ads violate this specification by placing all of the parameters without a preceding '?', so the feature evaluates to false for these examples.

- Size and dimensions in URL: Ad URLs very often contain information about the size of the ad it should display or the dimensions of the screen or browser it is going to be displayed on. One feature is if the URL contains an ad size. The format of the ad size we checked for is 2-4 numeric digits followed by the character x and then again followed by 2-4 numeric digits. Some ads contain the screen or browser dimensions so this second feature checks for the presence of one or more keywords from: screenheight, screenwidth, browserheight, browserwidth, screendensity, screenresolution and browsertimeoffset.

All features used are binary . More feature extraction and selection techniques can be employed for better performance of classifier.Infact having real valued features along with binary features allows all features to be given equal weight at training time.

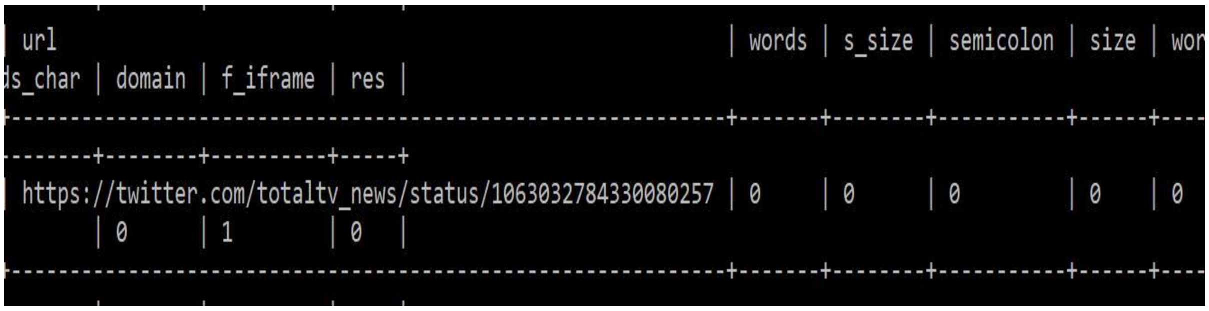


SAMPLE DATASET



```
| url                                                                | words | s_size | semicolon |
 size | words_char | domain | f_iframe | res |
+--------------------------------------------------------------------+-------+--------+----------+
------+------------+--------+----------+-----+
| https://tpc.googlesyndication.com/safeframe/1-0-32/html/container.html | 1     | 0      | 0        |
 0    | 1          | 0      | 1        | 1   |
| https://tpc.googlesyndication.com/safeframe/1-0-32/html/container.html | 1     | 0      | 0        |
 0    | 1          | 0      | 1        | 1   |
+--------------------------------------------------------------------+-------+--------+----------+
```

AD based data point



```
| url                                                      | words | s_size | semicolon | size | wor
ds_char | domain | f_iframe | res |
+----------------------------------------------------------+-------+--------+-----------+------+---
--------+--------+----------+-----+
| https://twitter.com/totaltv_news/status/1063032784330080257 | 0     | 0      | 0         | 0    | 0
        | 0      | 1        | 0   |
+----------------------------------------------------------+-------+--------+-----------+------+---
```

NON AD based data point

7.0 EVALUATION METHODOLOGY & RESULTS

As soon as we end up collecting data and extract its corresponding features , we save the dataset in our database .Then we export dataset into csv file using mysql workbench[9]

To determine what classification scheme fit our data the best, we implemented several of the most common classifiers used for supervised learning. To implement the classification, we made use of the machine learning development kit in Python called Scikit-Learn.

We split our dataset in 80%train dataset and 20% dataset for testing purposes.
But later train our model for whole data and test data would be the new data point(url's feature vector).

Now we train the model on our data locally and save the model in flask server using keyword "pickle". (REFER appendix D)[6]

Average Accuracy achieved (all features):

Classifier Algorithm	Avg. Percentage Accuracy
LOGISTIC REGRESSION	90.89
SVM	92.643

Implement all classification algorithms (suitable for binary classification)to check which scheme fits our training data best to give best results.

8.0 CONCLUSION

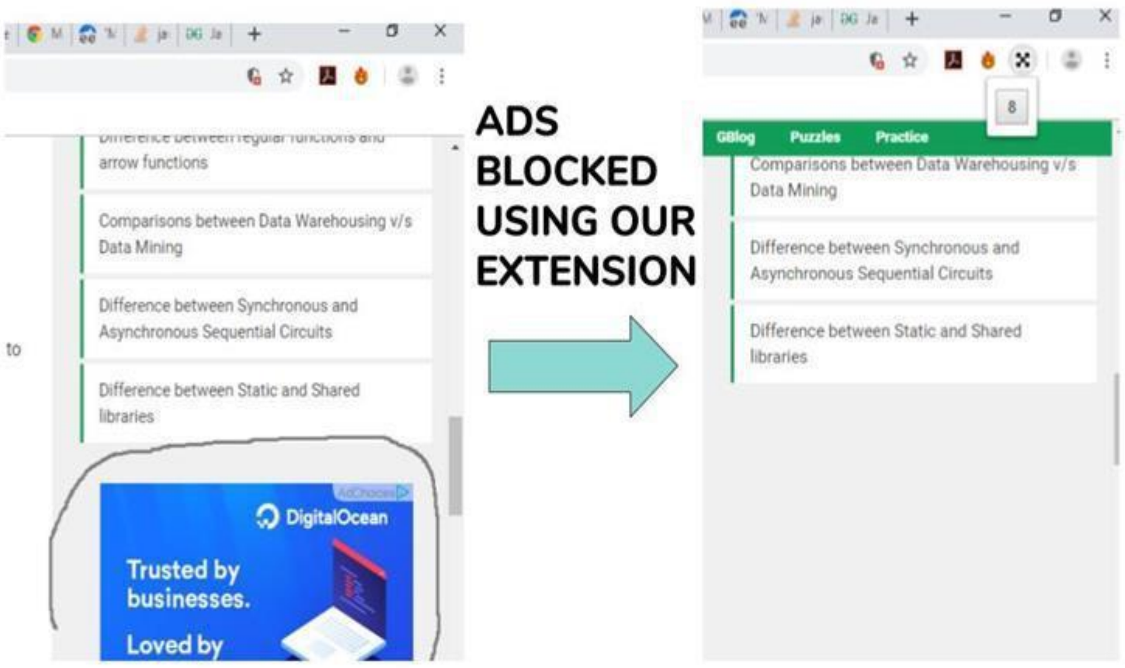
While the traditional ad blocking model has been successful and efficient since its creation in 2002, the manual efforts behind the scenes might be better placed elsewhere. Undesirable resources that could be irritating, privacy-violating or maliciously loaded as ads, need a more automatic way to be detected in the future. The landscape of ads is continuously changing either to embody more security threats or simply to combat ad blockers.

We have developed a machine learning based classifier for ads which was able to automatically learn currently known ads with a high accuracy and upto 50% of new ads that would otherwise necessitate manual filter creation. Our classifier was also able to detect a large number of ad and tracking URLs that current filter lists aren't able to identify, either due to their lack of defining URL tokens from which filters can be constructed or because the filters matching these haven't been manually identified yet.

We believe that this line of research has the potential to further enable user choice regarding what ads, tracking beacons, and other undesirable web assets are loaded on their machines, improving several aspects of the experience and web security for web users.

Sample Output:

The screenshots of the sample ad blocked are :


ADS BLOCKED USING OUR EXTENSION
arrow functions
Comparisons between Data Warehousing v/s Data Mining
Difference between Synchronous and Asynchronous Sequential Circuits
Difference between Static and Shared libraries
DigitalOcean
Trusted by businesses.
Loved by
GBlog
Puzzles
Practice
8

REFERENCES

1. https://developers.chrome.com/extensions
2. http://flask.pocoo.org/
3. https://dev.mysql.com/doc/
4. https://docs.python.org/
5. https://arxiv.org/pdf/1805.09155.pdf (other approach)
6. https://docs.python.org/3/library/pickle.html
7. https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html
8. https://www.w3schools.com/js/js_json_stringify.asp
9. https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-table.html.
10. https://www.w3schools.com/jsref/event_onload.asp
11. https://www.w3schools.com/python/python_pip.asp

A.0: CONTENT SCRIPT

client.js

```
window.onload=function() //inject content script when DOM is ready
{
       //chrome.runtime.onMessage.addListener(call); //Use to communicate with
       //function call(request, sender, sendResponse) //background script


       //try
       /*var lookup=document.getElementsByTagName("a"); // Extracting all <a> tags from
       for(var i=0;i<lookup.length;i++)                // document
       {
              //console.log(lookup[i]);
              if(lookup[i].getAttribute("href")!=null )
              {
                     var x="0";
                     func(lookup[i],x);
              }
              //var look=lookup[i].getElementsByTagName("iframe")
              //find(look)
       }*/


       var look_up=document.getElementsByTagName("iframe");
       find(look_up);

function find(look)//Extracting all <a> tags from an iframe recursively.
{
       for(var i=0;i<look.length;i++)
       {
              if((look[i].src)!=null && look[ i ].src.length > 0 )
              {
                            //console.log("ASk",look[i].src);
                     var data={"url":"","domain":"","iframe":""};
                     data.url=(look[i].src);
                     data.domain=window.location.href;
                     data.iframe="1";
                     var myjson=JSON.stringify(data);
                     //req(myjson);
                     pred(myjson,look[i]);

              }
              if(look[i].contentDocument!=null)
              {
              Var look1=look[i].contentWindow.document.getElementsByTagName("a");
                     for(var j=0;j<look1.length;j++)
                     {
                            if(look1[j].getAttribute("href")!=null )
                            {
                                   var y="1";
                            if(look1[j].getAttribute("href")!="javascript:void(0)" )
                                          func(look1[j],y);
                            }

                     }
                 Var
look2=look[i].contentWindow.document.getElementsByTagName("iframe");
                            console.log("ASSSSSS",look2.length);
                            find(look2);
              }
       }
}



function req(data) //Sending data to flask using AJAX for storing
{
       var xhttp = new XMLHttpRequest();
         xhttp.onreadystatechange = function() {
           if (this.readyState == 4 && this.status == 200)
           { console.log("done");}
         };
         xhttp.open("POST", "http://IP:5000/store", true);
         xhttp.setRequestHeader("Content-type",
         "application/json/text"); xhttp.send(data);
}

var count=0;

function pred(data,elem)//
Sending data to flask using AJAX for prediction    {
       var xhttp = new XMLHttpRequest();
         xhttp.onreadystatechange = function()
        {
           if (this.readyState == 4 && this.status == 200)
              {
                     if(this.responseText == "[1]")
                     {
                            //hiding html element in case prediction is positive.
                                   count++;

                                   elem.style.display = "none";


       chrome.runtime.onMessage.addListener(function (msg, sender, sendResponse) {
                                   // First, validate the message's structure
              if ((msg.from === 'popup') && (msg.subject === 'DOMInfo'))
                            {

                                   var domInfo={"total":""};
                                   domInfo.total=count;
                                   sendResponse(domInfo);
                             }});
                     }
              }

         };
         xhttp.open("POST", "https://IP:5000/predict", true);
         xhttp.setRequestHeader("Content-type",
         "application/json/text"); xhttp.send(data);
}

function func(elem,flag)//Used to store data point in json and stringify
it. {
       var data={"url":"","domain":"","iframe":""};
       data.url=(elem.getAttribute("href"));
       data.domain=window.location.href;
       data.iframe=flag;
       var myjson=JSON.stringify(data);

       pred(myjson,elem);
}

}//in xhttp.open() write IP address of machine where flask is running.
```

A.1: SERVER SCRIPT

dbconn.py

```
from flask import Flask, request
import json
import pymysql
#feature extractor code is imported
import feature_extractor as feature
import re
from urlparse2 import urlparse
import pickle
import numpy as np
app = Flask(__name__)


def func(d):
        db = pymysql.connect("localhost","root","bhalomaa","project" )
        cursor = db.cursor()
        #insertStatement = "INSERT INTO sam (name) VALUES (d["url"])"
        #print(type(d))
        val=str(d["url"])
        base_url=str(d["domain"])
        from_iframe=str(d["iframe"])
        token=re.split('\W+',val)
        o = urlparse(val)
        x=o.params+o.query
        #Extract features and save data into mysql database.
        a=feature.check_words(token)
        b=feature.check_screen_size(token)
        c=feature.check_semicolon(x)
        da=feature.check_size(x)
        e=feature.check_words_char(val)
        f=feature.check_domain(val,base_url)
        cursor.execute("INSERT INTO
tem(url,words,s_size,semicolon,size,words_char,domain,f_iframe)
VALUES (%s,%s,%s,%s,%s,%s,%s,%s)",(val,a,b,c,da,e,f,from_iframe))
        db.commit()
        db.close()


def func1(d):
        val=str(d["url"])
        base_url=str(d["domain"])
        from_iframe=str(d["iframe"])
        token=re.split('\W+',val)
        o = urlparse(val)
        x=o.params+o.query

        a=feature.check_words(token)
        b=feature.check_screen_size(token)
        c=feature.check_semicolon(x)
        da=feature.check_size(x)
        e=feature.check_words_char(val)
        f=feature.check_domain(val,base_url)
        x=[[a,b,c,da,e,f,from_iframe]]
        filename='finalyy_model.sav'
        #prediction using saved model in the server
        loaded_model = pickle.load(open(filename, 'rb'))
        predicted = loaded_model.predict(x)
        y=np.array2string(predicted)
        #p=predicted.tostring()
        #pp=np.fromstring(p)
        #print (type(y),y)
        return y

#used when content script wants server to store data
@app.route('/store', methods = ['POST'])
def store():

        da=request.data
        #print json(da)
        d=json.loads(da)
        func(d)

        return 'ok'

#used when content script wants server to predict output
@app.route('/predict', methods = ['POST'])
def predict():

        da=request.data
        #print json(da)
        d=json.loads(da)
        res=func1(d)
        #if(res=="[1]"):
        return res

if __name__ == '__main__':
        # run!
        #app.run()
        app.run(host='0.0.0.0',ssl_context='adhoc')
```

A.2: FEATURE EXTRACTOR

Feature_extractor.py

```
import re
from urlparse2 import urlparse

def check_words(tokens_words):

   bag=['ad', 'advert', 'popup', 'banner', 'sponsor', 'iframe','googlead',
'adsys', 'adser','adclick','googlesyndication','adroll','adsettings']
   for ele in bag:
      if(ele in tokens_words):
         return 1

           for ele in bag:
                     for el in tokens_words:
                               if(ele in el):
                                         #print el
                                         #print ele
                                         return 1

   return 0
def check_words_char(url):
           bag=['ad', 'advert', 'popup', 'banner', 'sponsor', 'iframe','googlead',
'adsys', 'adser','adclick','googlesyndication','adroll','adsettings']
           symb=['.','/','&','=',';','-','_']
           for ele in bag:
                     for ele1 in symb:
                               #pattern = re.compile("^.*%s%s.*$"%ele%ele1)
                               #print(ele)
                               #print(ele1)
                               #print(url)
                               if(re.search(r".*"+re.escape(ele)+re.escape(ele1)+r".*",url)):
                                         #print ele1
                                         return 1
           return 0

def check_screen_size(tokens_words):

           bag=['screenheight', 'screenwidth','browserheight', 'browserwidth', 'screendensity',
'screenresolution' , 'browsertimeoffset']
           for ele in bag:
                     if(ele in tokens_words):
                               return 1

           for ele in bag:
```

```
        for el in tokens_words:
            if(ele in el):
                return 1

    return 0

def check_semicolon(x):
    y=";"
    if(y in x):
        return 1

    return 0

def check_size(x):
    pattern = re.compile("^.*([0-9]{2,4}x[0-9]{2,4}).*$")
    if(re.search(pattern,x)):
        return 1
    else:
        return 0

def check_domain(url,base_url):
    y=urlparse(url)
    z=urlparse(base_url)
    if(y.netloc==z.netloc):
        return 1

    return 0
```

A.3 : PARSER

parser.py

```
from adblockparser import AdblockRules
import pymysql

#converting easylist of adblock plus to list for matching data points against it.
lines=[]
with open("easylist.txt") as file:
        for line in file:
                line=line.strip()
                lines.append(line)


#set connection with mysql database. Each data point is matched against the above list to label it as
# "1" (ad based data) or  "0"  (non ad based data) and finally update mysql Db.

db = pymysql.connect("localhost","root","bhalomaa","project" )
cursor = db.cursor()
cursor.execute("select * from tem")
for row in cursor.fetchall():
        x=row[0]
        #cur = db.cursor()
        #print x
        if rules.should_block(row[0])==True:
                #sql='INSERT INTO pro1(res) VALUES ('+'"'+"1"+'"'+')'

                sql='UPDATE tem SET res=%s where url=%s'
                cursor.execute(sql,("1",x))
        else:
                #sql='INSERT INTO pro1(res) VALUES ('+'"'+"0"+'"'+')'
                sql='UPDATE tem SET res=%s where url=%s'
                cursor.execute(sql,("0",x))
db.commit()
db.close()
```

A.4 : TRAINER

trainer.py

```
print "Content-type:text/html\r\n\r\n"
import pandas as pd
from sklearn import *
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.cross_validation import train_test_split
from bs4 import BeautifulSoup import scipy

import nltk
from sklearn import svm
from sklearn.linear_model import
LogisticRegression import pickle

#reading data from csv to dataframe.
names = ['words', 's_size', 'semicolon','size','words_char','domain', 'f_iframe', 'res']
df=pd.read_csv('check_data.csv', error_bad_lines=False,engine='python')
#,names=names)
#d=pd.DataFrame(df)

#separating feature vector from labelled result.
d=df.values
y=d[:,8]
#print d
y=y.astype('int')
x=d[:,1:8]

#(splitting data in 80-20 % of train & test data respectively). Eventually to be trained using whole
#data.
xtrain, xtest,ytrain,ytest = train_test_split(x,y, test_size=0.2)



#SVM/Logistic regression is used . More algos can be used.
model = svm.SVC(kernel='linear', C = 1.0)
#model = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomial').fit(xtrain,
ytrain)
model.fit(xtrain,ytrain)
#Model is saved using pickle in the same directory.
filename = 'finalyy_model.sav'
pickle.dump(model, open(filename, 'wb'))
predicted = model.predict(xtest)
print type(predicted)
print predicted,ytest
print model.score(xtest,ytest)
```

APPENDIX F: CHROME EXTENSION

To get started with chrome extensions follow the link.
(https://developer.chrome.com/extensions/getstarted).

manifest.json: Every component of extension needs to get registered at

```
manifest.json {
    "name": "ADVERTISEMENT BLOCK",
    "version": "1.0",
    "manifest_version": 2,
    "description": "Block ADS using ML!",
    "permissions": ["storage"],
   "content_scripts": [
   {
     "matches": ["<all_urls>"],

    "js": ["client.js"]
   }
 ],
    "browser_action":
 {
           "default_icon": "adv.png" ,
            "default_popup": "popup.html"
    }
  }
```

Client.js

```
window.onload=function()
{
	//chrome.runtime.onMessage.addListener(call);

	//function call(request, sender, sendResponse)


	//try
	/*var
	lookup=document.getElementsByTagName("a");
	for(var i=0;i<lookup.length;i++) {
		//console.log(lookup[i]);
		if(lookup[i].getAttribute("href")!=null )
		{
			var x="0";
			func(lookup[i],x);
		}
		//var look=lookup[i].getElementsByTagName("iframe")
		//find(look)
	}*/


	var look_up=document.getElementsByTagName("iframe");
	find(look_up);



function find(look)
{
	for(var i=0;i<look.length;i++)
	{
		if((look[i].src)!=null && look[ i ].src.length > 0 )
		{
				//console.log("ASk",look[i].src);
			var data={"url":"","domain":"","iframe":""};
			data.url=(look[i].src);
			data.domain=window.location.href;
			data.iframe="1";
			var myjson=JSON.stringify(data);
			//req(myjson);
			pred(myjson,look[i]);

		}
		if(look[i].contentDocument!=null)
		{
		Var look1=look[i].contentWindow.document.getElementsByTagName("a");
			for(var j=0;j<look1.length;j++)
			{
				if(look1[j].getAttribute("href")!=null )
				{
					var y="1";
				if(look1[j].getAttribute("href")!="javascript:void(0)" )
						func(look1[j],y);
				}

			}
		    Var
look2=look[i].contentWindow.document.getElementsByTagName("iframe");
				console.log("ASSSSSS",look2.length);
				find(look2);
		}
	}
}
function req(data)
{
      var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
          if (this.readyState == 4 && this.status == 200)
          { console.log("done");
             }
        };
        xhttp.open("POST", "http://localhost:5000/store", true);
        xhttp.setRequestHeader("Content-type",
        "application/json/text"); xhttp.send(data);
}

var count=0;

function pred(data,elem)
{
      var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function()
       {
          if (this.readyState == 4 && this.status == 200)
             {
                    if(this.responseText == "[1]")
                    {

                                elem.style.display = "none";
                                //elem.style.visibility = 'hidden';
                                count++;
                                console.log(count);


      chrome.runtime.onMessage.addListener(function (msg, sender, sendResponse) {
                                // First, validate the message's structure
             if ((msg.from === 'popup') && (msg.subject === 'DOMInfo'))
                          {

                                var domInfo={"total":""};
                                domInfo.total=count;
                                sendResponse(domInfo);
                           }});

                    }



          console.log(elem,this.responseText);
             }

        };
        xhttp.open("POST", "https://172.18.3.37:5000/predict", true);
        xhttp.setRequestHeader("Content-type",
        "application/json/text"); xhttp.send(data);
}

function func(elem,flag)
{
      var data={"url":"","domain":"","iframe":""};
      data.url=(elem.getAttribute("href"));
      data.domain=window.location.href;
      data.iframe=flag;
      //console.log(data.url);
      var myjson=JSON.stringify(data);
      //console.log(myjson);
      //req(myjson);
      pred(myjson,elem);
}

}
```

Adv.png : The extension icon to be used.

Background.js : use in case we don’t want content script to get injected automatically or when we want message passing between content script(client.js) and background script.

```
chrome.browserAction.onClicked.addListener(clicked);
console.log("hello");

function clicked()
{
	chrome.tabs.query({active: true, currentWindow: true}, send);
	function send(tabs) {

	let msg=
	{
		txt:"hello world",
		//mail:"abcdefg@xyz.com"
	}
	chrome.tabs.sendMessage(tabs[0].id, msg);
	}
}
```

popup.html & popup.js : helps to generate user interface.

Popup.html

```
<!DOCTYPE html>
 <html>
  <head>
   <style>
    button {
     height: 30px;
     width: 30px;
     outline: none;
    }
   </style>

  </head>
  <body>
   <button id="cz"></button>
        <script type="text/javascript" src="popup.js"></script>
  </body>

 </html>
```

Popup.js: In our case we are trying to display number of ads blocked in the current website

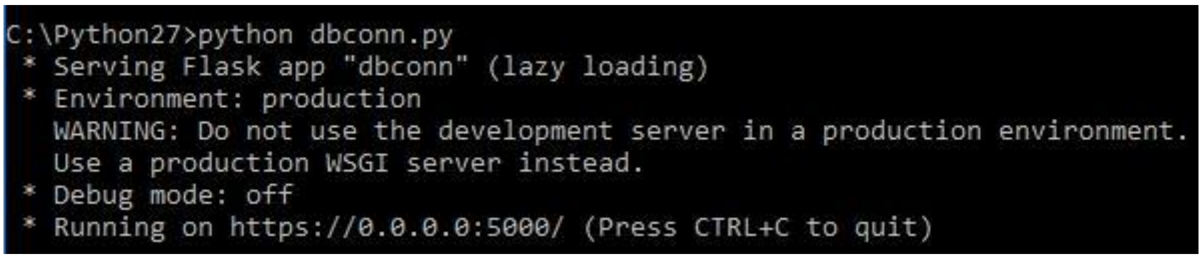
```
// Once the DOM is ready...
window.addEventListener('DOMContentLoaded', function ()
 { chrome.tabs.query({
  active: true,
  currentWindow: true },
 function (tabs) {
 chrome.tabs.sendMessage(
    tabs[0].id,
    {from: 'popup', subject: 'DOMInfo'},
              function (info) {

                     //body.onload=function(){
                            if (typeof info === "undefined") {
                                   console.log('a');
                            }
                            else{

                            document.getElementById('cz').innerHTML= info.total;}
              });
    // ...also specifying a callback to be called
    //   from the receiving end (content
    script) //setDOMInfo(info));
 });
});
```

APPENDIX B: BUILD & RUN

STEP I : Run the flask server on your machine. Type python dbconn.py in command prompt in the same directory where dbconn.py is kept.



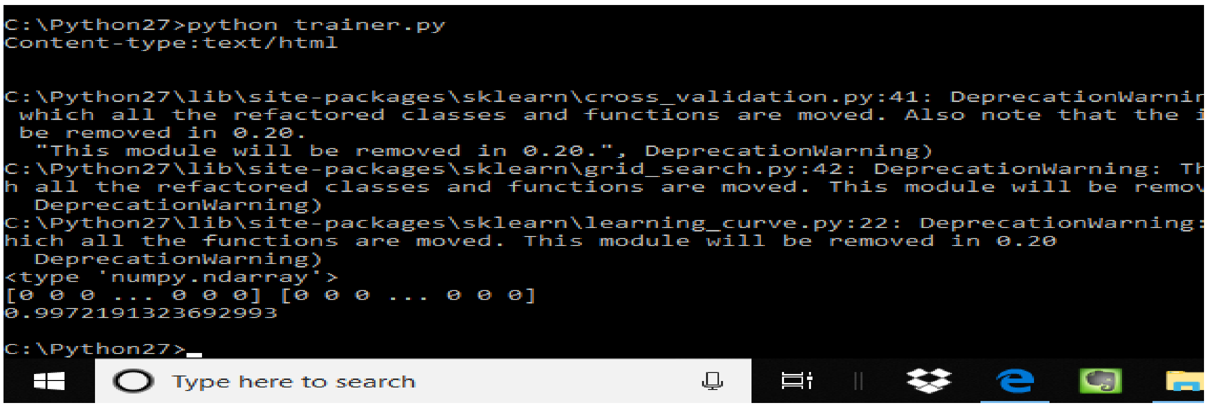
```
C:\Python27>python dbconn.py
 * Serving Flask app "dbconn" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on https://0.0.0.0:5000/ (Press CTRL+C to quit)
```

STEP II : First collect lot of data by installing the extension on your browser using req() function .Features will be extracted through same script(dbconn.py) and saved in mysql db.

STEP II : Then label the data by typing “python parser.py”.

STEP III : Then import the data to .csv format using mysql workbench.

STEP IV : Then train the model by typing “python trainer.py” and save the model back in server.



```
C:\Python27>python trainer.py
Content-type:text/html


C:\Python27\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarnin
 which all the refactored classes and functions are moved. Also note that the i
 be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Python27\lib\site-packages\sklearn\grid_search.py:42: DeprecationWarning: Th
h all the refactored classes and functions are moved. This module will be remov
  DeprecationWarning)
C:\Python27\lib\site-packages\sklearn\learning_curve.py:22: DeprecationWarning:
hich all the functions are moved. This module will be removed in 0.20
  DeprecationWarning)
<type 'numpy.ndarray'>
[0 0 0 ... 0 0 0] [0 0 0 ... 0 0 0]
0.9972191323692993

C:\Python27>_
```

STEP V : Then again run dbconn.py but use pred() function which will help predicting ad and subsequently block it.

APPENDIX C: SYSTEM DESCRIPTION

- Windows 10 Home(64 bit OS, 8 GB RAM) is used.
- Google chrome (Version 74.0.3729.157 (Official Build) (64-bit)) is used.
- Python 2.7 with pip to be used in windows.(libraries are installed using "pip install x" command).[11]
- MySQL Server 5.7, MySQL Workbench 6.3CE and MySQL Shell 1.0.11 should be installed.
- Flask(1.0.2 version) should be configured.