Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Master's thesis

Application of machine learning methods to solve the problem of user identification on various digital devices

Bc. Elena Ivanova

Supervisor: Ing. Karel Frajták, PhD

Study Program: Open Informatics

Field of Study: Software Engineering

May 24, 2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Ivanova** Jméno: **Elena** Osobní číslo: **474700**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Application of machine learning methods to solve the problem of user identification on various digital devices

Název diplomové práce anglicky:

Application of machine learning methods to solve the problem of user identification on various digital devices

Pokyny pro vypracování:

The task of recognizing one user across multiple devices became extremly important nowadays as everyone owns different devices to perform tasks and user's identity becomes fragmented which isn't good for advertising.
The diploma's main goal is to develop a system which could predict whether two cookies belongs to one user or not.
The main challenges are: which machine learning model to choose, how to deal with data incompleteness and excess of parameters et. al.

Seznam doporučené literatury:

1)M. Landry, S. R. S and R. Chong, ""Multi-layer Classification: ICDM 2015 Drawbridge Cross-Device Connections Competition,"" 2015 IEEE International Conference on Data Mining Workshop (ICDMW), Atlantic City, NJ, 2015.
2)T. R. Anand and O. Renov, ""Machine Learning Approach to Identify Users Across Their Digital Devices,"" 2015 IEEE International Conference on Data Mining Workshop (ICDMW), Atlantic City, NJ, 2015.
3)J. Walthers, ""Learning to Rank for Cross-Device Identification,"" 2015 IEEE International Conference on Data Mining Workshop (ICDMW), Atlantic City, NJ, 2015.
4)M. S. Kim, J. Liu, X. Wang and W. Yang, ""Connecting Devices to Cookies via Filtering, Feature Engineering, and Boosting,"" 2015 IEEE International Conference on Data Mining Workshop (ICDMW), Atlantic City, NJ, 2015.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Karel Frajták, Ph.D., katedra počítačů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **16.04.2018** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce: **30.09.2019**

Ing. Karel Frajták, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studentky

Acknowledgements

I would like to express my sincere gratitude to my KFU supervisor, Kamil Khadiev for his patient guidance and immense to help on my thesis. I want to specially thank Ilya Bokarev for supporting and encouraging me in a way that no other ever could. Finally, thanks a lot to my family for their faith in me.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.

I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

Kazan, May, 2018 ________________________

Abstract

The task of recognizing one user across multiple devices became extremely important nowadays as everyone owns different devices to perform tasks and user's identity becomes fragmented which isn't good for advertising.

The diploma's main goal is to develop an algorithm which could link computers and mobile devices that belong to the same person. We are provided with anonymous data that include user’s behavior on sites and mobile apps along with visited IP addresses.

The thesis is focused on various machine learning techniques that could be applied to solve the problem.

Contents

List of Figures

List of Tables

Introduction

Nowadays, with constant development of technology, the number of our devices for accessing the Internet is increasing rapidly. Every day we use mobile phones, tablets and laptops for googling and watching videos with cats. If all devices are not joined by a shared account, then our personality becomes fragmented. Services can’t offer the most appropriate advertising or relevant search results. Personalization is blocked by the inability to realize that several users on different devices are the same person.

Returning integrity to the user is especially important in marketing. There is a special advertising direction describing how to target the same user across multiple devices which is called "cross-device targeting" or "cross-device marketing". This direction is extremely popular. For example, Google added the ability to identify users across devices to Google Analytics in 2017 [1], there are also hundreds of large companies that generate business solutions in the area ([2], [3], [4]).

Why is it so important for marketing to tie devices together? The thing is that devices of different types receive different amounts of information. One of the reasons why mobile advertising is not in high demand among marketers is the lack of a large amount of data. In the "big" Internet, thanks to cookies we know almost everything about users - which sites they visited, what they were looking for, what they are interested in. On phones and tablets we usually know only device’s type and operator.

Goal

The main goal of this work is the construction of an algorithm that can find and link mobile devices and personal computers (cookies) belonging to one user. The algorithm will use the machine learning methods to achieve the maximum accuracy of predictions. Particular attention will be paid to the analysis of data and work with them, as well as the choice of model for training.

In the course of the work, the following tasks will be solved: how to work with large data effectively and how to reduce them; how to classify the problem and how the chosen approach changes preferred methods for solution. In particular, our algorithm of choice for the competition is a gradient boosting machine which produces a score by iteratively fitting small decision trees. The problem itself will be considered as a binary classification problem or as a ranking task. To improve the result obtained, various modifications of the learning process will be applied such as: bagging, selection of parameters using cross-validation, etc.

The formulation of the problem together with the data was taken from the international scientific and practical contest Kaggle [5].The competition was organized by one of the industry leaders, Drawbridge [4], in 2015 and 340 teams took part in it.

1 Problem formulation

In this chapter we will strictly formulate the problem and how to evaluate the results. We also provide a brief description of the provided tables.

The main task of the work is to build an algorithm for linking mobile devices and cookies based on anonymous information and public data, such as user-visited IP addresses, sites and mobile applications. The solution of the problem should be received in the following form: for each device it is necessary to get the list of cookies that belong to the same user as devices.

The correctness of the provided list is evaluated based on the mean score calculated by the formula (for more details see the Definitions):

where

– true positive decisions,
false positive decisions,
false negative decisions.

Data includes the following tables:

1. Device basic information table (dev_train_basic.csv and dev_test_basic.csv)

Property name	Description
Drawbridge Handle	Drawbridge identifier, uniquely identify a person behind device and cookie. The owner is equal to -1 for the test set.
Device ID	Unique device identifier
Device type	Device type (categorical)
Device OS version	Device OS version (categorical)

Device Country Info	Which country this device belongs to (categorical)
Anonymous_c0	Anonymous feature to describe device (Boolean)
Anonymous_c1	Anonymous feature to describe device (categorical)
Anonymous_c2	Anonymous feature to describe device (categorical)
Anonymous_5	Anonymous feature to describe device
Anonymous_6	Anonymous feature to describe device
Anonymous_7	Anonymous feature to describe device

Table 1.1: Device basic information table

2. Cookie basic information table (cookie_basic.csv)

Property name	Description
Drawbridge Handle	Drawbridge identifier, uniquely identify a person behind device and cookie. The owner is equal to -1 for the test set.
Cookie ID	Unique cookie identifier
Computer OS type	Computer OS version (categorical)
Browser version	Cookie browser version (categorical)
Cookie country info	Which country this cookie belongs to (categorical)
Anonymous_c0	Anonymous feature to describe cookie (with the same meaning as feature Anonymous_c0 from dev_train_basic)

Anonymous_c1	Anonymous feature to describe cookie (with the same meaning as feature Anonymous_c1 from dev_train_basic)
Anonymous_c2	Anonymous feature to describe cookie (with the same meaning as feature Anonymous_c2 from dev_train_basic)
Anonymous_5	Anonymous feature to describe cookie (with the same meaning as feature Anonymous_5 from dev_train_basic)
Anonymous_6	Anonymous feature to describe cookie (with the same meaning as feature Anonymous_6 from dev_train_basic)
Anonymous_7	Anonymous feature to describe cookie (with the same meaning as feature Anonymous_7 from dev_train_basic)

Table 1.2: Cookie basic information table

3. IP table (id_all_ip.csv) describes the joint behavior of device or cookie on IP address.

Property name	Description
Device/cookie ID	Device or cookie identifier
Device or Cookie	Boolean variable. If its equal to 0 then the table row refers to the device, otherwise – to the cookie
IP	IP address
Freq count	How many times have we seen dev or cookie in column 1 appear on the IP in column 3

Anonymous Count 1	Anonymous number that describes the behavior
Anonymous Count 2	Anonymous number that describes the behavior
Anonymous Count 3	Anonymous number that describes the behavior
Anonymous Count 4	Anonymous number that describes the behavior
Anonymous Count 5	Anonymous number that describes the behavior

Table 1.3: IP table

4. IP aggregation table (ipagg_all.csv) provides aggregated behavior of each IP.

Property name	Description
IP	IP address
Is cell IP	If IP is cellular IP or not. 1 for cellular and 0 for non-cellular.
Total Freq	Total number of observations seen on this IP (This number is the aggregated observation count on all the devices and cookies seen from this IP)
Anonymous count c0	Anonymous count that describes the behavior of the IP
Anonymous count c1	Anonymous count that describes the behavior of the IP
Anonymous count c2	Anonymous count that describes the behavior of the IP

Table 1.4: IP aggregation table

5. Property observation table (id_all_property.csv) provides the information regarding website (for cookie) and mobile app (for device) that user has visited before.

Property name	Description
Device/cookie ID	Device or cookie identifier
Device or cookie indicator	Boolean variable. If its equal to 0 then the table row refers to the device, otherwise – to the cookie
Property ID	Website name for cookie, and mobile app name for the device
Property unique count	How many times have we seen device or cookie on this property

Table 1.5: Property observation table

6. Property category table (property_category.csv) lists the categorical information of the website/mobile app.

Property name	Description
Property ID	Website name for cookie, and mobile app name for the device
Property category	Category of the website or the mobile app

Table 1.6: Property category table

3 Background

In this chapter, we will provide formal definitions of terms and methods that would be used further. Some important notes will be given about F-score and other classification measures. We will have a deep look into gradient boosting and how it can be applied to classification problem.

3.1 Definitions

Classification problem.

Let X be a set of descriptions of objects, Y – a finite set of numbers (names, labels) of classes. There is an unknown target dependence – mapping y^*: $X \to Y$ whose values are known only at the objects of the final training set $X^m = \{(x_1,y_1),\cdots,(x_m,y_m)\}$. The task is to construct an algorithm a: $X \to Y$ which is able to classify a [6]n arbitrary object $x \in X$.

Often, instead of the previous one, a probabilistic statement of the classification problem is used:

Let the set of pairs be a probability space with an unknown probability measure . There is a finite training set of observations generated according to the probability measure . It is required to construct an algorithm that can classify an arbitrary object .

If we speak of binary classification, then the number of classes is two.

Learning to rank problem.

Let X be the set of descriptions of objects, be a training set, be a regular order on the pairs .

It is required to construct a ranking function such that

There are 3 approaches to the solution of the learning to rank problem:

1. Point-wise
   In this case, it is assumed that each query-document pair in the training data has a numerical or ordinal score. Then the learning-to-rank problem can be approximated by a regression problem – given a single query-document pair, predict its score.

2. Pairwise
   In this case, the learning-to-rank problem is approximated by a classification problem – learning a binary classifier that can tell which document is better in a given pair of documents. The goal is to minimize the average number of inversions in ranking.

3. List-wise
   It consists in constructing a model, the input of which is received immediately by all the documents corresponding to the query, and the output is obtained by their permutation. Adjustment of model parameters is carried out for direct maximization of one of the above ranking metrics.

Decision trees

Decision Tree is a decision support tool used in statistics and data analysis for predictive models. The structure of the tree includes nodes and branches. Each inner node ask some question based on the attributes of the model, each leaf keeps value of the objective function. To get a solution, you need to go down the tree to the leaf and take the value that it keeps.

The goal is to create a tree that predicts the value of the target variable based on several variables at the input.

To solve classification problems, the model is adapted as follows: class label is written in the leaf of the tree and each element that gets in the leaf automatically refers to this class.

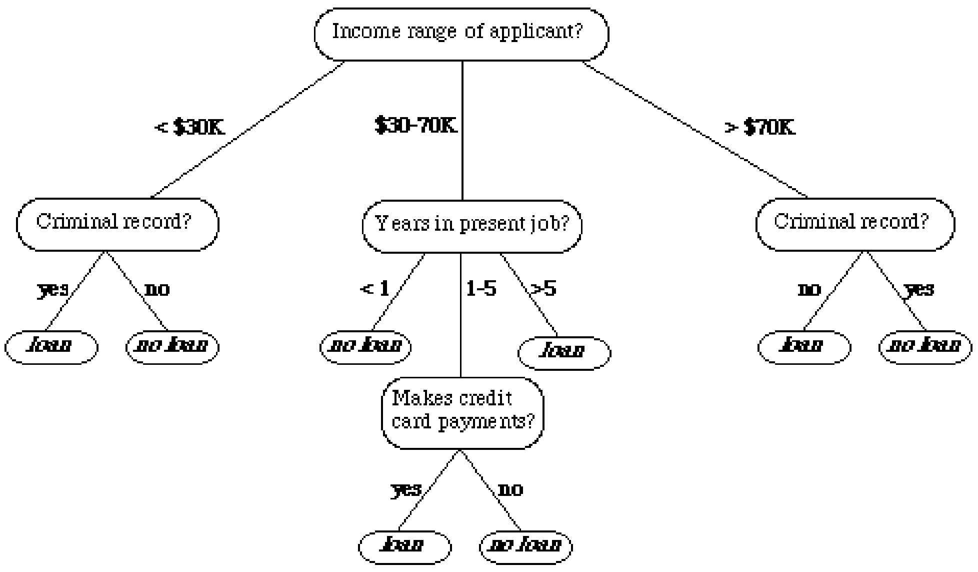

Figure 3.1: Example of decision tree

The decision tree is built on the basis of the training set so as to minimize the error in each leaf. Unfortunately, the creation of the specified tree is an NP-complete problem [7], therefore when building the decision tree, greedy algorithms are used. They do not guarantee that the constructed tree will be optimal.

Decision trees are used as an inner part of more complex algorithms: bagging, gradient boosting, Random forest, etc [8].

Bagging

Bagging (Bootstrap aggregation) is a classification technique that uses algorithm compositions, each of which is learned independently. The result of the classification is determined by voting or averaging. It is expected that the result of the forecast of the aggregated classifier will be much more accurate than the result of the forecast of a single model on the same data set.

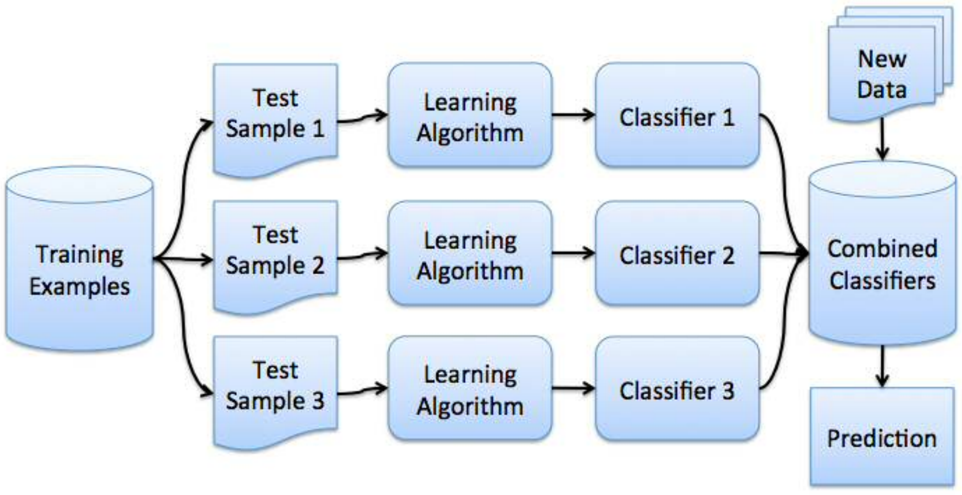

Figure 3.2: Basic principle of bagging

Statistical measures of a binary classification

Suppose that we have two classes and an algorithm that predicts the belonging of each object to one of the classes. The following table calls confusion matrix:

	True Positive (TP)	False Positive (FP)
	False Negative (FN)	True Negative (TN)

Table 3.1: Confusion matrix

Here   is the algorithm response on the object, and   is the true class label on the same object. Thus, there are two types of classification errors: false-negative and false-positive.

Based on the confusion matrix, the simplest metrics are determined to assess the quality of the classification:

1. Precision shows the proportion of objects that are called positive by the classifier and are in fact positive. It is calculated by the formula:

2. Recall determines which fraction of objects of a positive class from all objects of a positive class found an algorithm. The formula for the calculation is as follows:

The difference between accuracy and completeness is clearly illustrated in the figure below.

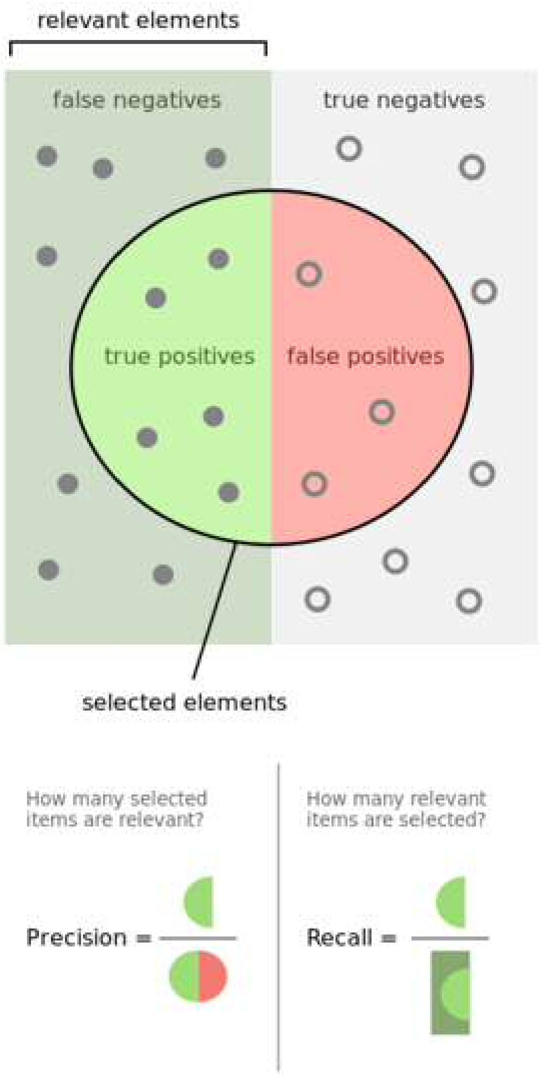

Figure 3.3: Visual representation of the basic metrics

It is clear that the higher the precision and recall, the better. But in real life, maximum precision and recall are not achievable at the same time and we have to look for a certain balance. F-score can give us this balance:

defines the weight of precision in the metric.

F-score is the harmonic mean with β = 1 (precision and recall are equally important), but if the parameter takes values in the range (0; 1), precision is given preference [9].

3.2 Gradient boosting [6]

Consider the problem of recognizing objects from a multidimensional space with a space of labels . Suppose we are given a training sample where . Also consider that we know the true values of the labels of each object where . It is necessary to build a recognizing operator that can predict the labels for each new object .

Suppose we are given a family of basic algorithms , each element of which is defined by some parameter vector .

We will search for the final classification algorithm in the form of a composition

Since the selection of the optimal set of parameters is a laborious task, we will try to construct such a composition by means of a greedy buildup, each time adding to the sum a term that is the most optimal algorithm possible. We assume that we have already constructed a classifier of length . Thus, the problem reduces to finding the pair of the most optimal parameters for the classifier of length :

The criterion of optimality is a loss function showing how much the predicted response differs from the correct answer . And then the error functional is minimized

It is remarkable that the error functional is a real function depending on the points in -dimensional space, and we need to solve the problem of minimizing this functional. We do this by implementing one step of the gradient descent method. As a point for which we will seek the optimal increment, consider . Let us find the gradient of the error functional:

Thus, by the method of gradient descent, it is most advantageous to add a new term to the classifier as follows:

where is chosen by linear search in real numbers R:

However, is only a vector of optimal values for each object , and not a basic algorithm from the family defined by . Therefore, we need to find most similar to . We do this, again minimizing the error functional, based on the principle of explicitly maximizing indentation:

which simply corresponds to the basic learning algorithm. Next, find the coefficient , using the linear search:

Gradient boosting is used to solve a wide range of problems [10]. Let's consider its features in the appendix to classification problems.

3.3 Gradient boosting in an application to classification problems

The idea of boosting is applicable to the classification problem. In the case of a binary classification, this means that . Then it is often assumed that each algorithm returns the real "degree" of the object's belonging to a certain class, and the resulting response is obtained by applying the threshold rule to the composition.

In the case of classification, the loss function from one argument is usually used:

in fact, indent is replaced by the product of the present class and the predicted value.

In this case, there is a slightly different view of the gradient boosting approach than the one described above. Under the gradient of the error functional, we can mean a vector of weights of training objects, multiplied by the correct values of classes:

where Then the learning algorithm in accordance with the principle of maximization of indentations acquires the following form:

Thus can be viewed from the point of view of the weights that are attached to objects and are taken into account when learning each basic algorithm.

The most commonly used loss functions are [11]:

1. Logistic loss

   This is the most common and often used loss function in binary classification.

2. Adaboost loss

Used in the classical implementation of the gradient descent algorithm Adaboost. Similar to Logistic loss, but it has a tougher exponential penalty for classification errors.

5 Design and Implementation

In this chapter we will define two main approaches to the problem. Implementation details will also be described here.

5.1 Methodology and Approach

Studying articles devoted to the competition, you can see that the task of cross-device connection is assigned by the authors to one of two types of tasks: binary classification (for example, in [12], [13], [14], [15], [16]) or learning to rank problem [17]. Let's consider each of the approaches.

Cross-device connection as a binary classification problem.

Let there be a device and a list of cookies , among which you want to select those whose owners are the same as the owner of the device. Then all pairs with information about devices and cookie make up the set (the set of descriptions of objects), and the set of labels , where 1 means that the pair has the same user, 0 - that users are different.

For each cookie from the list of candidates, the probability is predicted that the owner is the same. After that on the basis of the probabilities obtained, it is decided which cookies fall into the final list. The easiest way is to select the cookie that showed the highest probability, but algorithms can act more sophisticated, for example, look at the distribution of these probabilities, etc.

Cross-device connection as learning-to-rank problem

We have the same formulation as in the previous approach, but now we are not interested in the prediction of the connection of individual pairs , but the rearrangement of all candidates so that the most probable ones turn out to be less than others in rank. In a sense, this is a generalization of the previous approach: again we estimate the probability of each candidate, but now we take into account the presence of competitors.

To solve the ranking problem any of the approaches described above can be applied: in this paper we used a pairwise approach. Since pairwise approach is approximated by a binary classification, it does not require strong changes in the data structure and code as compared to the first formulation.

Conclusion on the analysis

Based on the results of the analysis, it was decided to implement both approaches and compare the effectiveness of models, training time and the complexity of implementation.

Gradient boosting was chosen as the main learning algorithm for both approaches due to its power in solving such problems.

5.2 Architecture of the system

The system consists of 4 logical parts:

1. DataBuilder. This module processes initial tables. It has a set of methods for loading information, building dataset and saving it after all.

2. ModelBuilder works with dataset constructed by DataBuilder. The module trains algorithms and do postprocessing.

3. Analytics module calculates coverage, sizes, creates plots etc.

4. Variables is a configuration file

5.3 Tools

Here is a list of the primary tools and frameworks which we used for solving cross-device connection problem. All used tools are free and open-source.

Used programming languages: Python 3.6.0 for all processing, training and data analysis procedures.

Python stack: scikit-learn, numpy libraries for data manipulation and model building; xgboost framework [18] for boosted trees algorithm; pickle – for storing and loading intermediate result. We also used re, csv and other Python packages.

Visualization: Matplotlib - Python libraries for dataset visualization.

Computation: Windows 10, Processor 2,4GHz Intel Core i7, RAM 16 GB.

IDE: PyCharm IDE

6 Experimental part

In this chapter, we will present the process of building the whole algorithm from the very beginning. The solution of the problem consisted of the following consecutive stages:

1. Preprocessing
   This stage included the merging of tables, downsampling and generation of new properties based on existing ones. The obtained set was divided into two unequal parts: 85% of all the rows were assigned to the training set, 15% to the validation set. The training involved only the first set of records, the second one was used to compare the overall efficiency of the algorithms.

2. Training of models and selection of parameters
   For each of the selected models, the parameters were iteratively selected, so as to minimize the error in the training sample. At the same step, the methods of bagging and composition of models were used.

3. Postprocessing
   During the stage we constructed an algorithm that received the desired output. Algorithm was based on the predictions of the models trained earlier. Joint model was built on this stage, too.

6.1 Preprocessing

One of the distinguishing features of the problem, singling it out against the background of other similar ones, was the absence of expected training set with rows and labels. All information was divided into several files and it was required to determine an algorithm that could join the tables and construct the training set.

Tables were joined according to the following scheme:

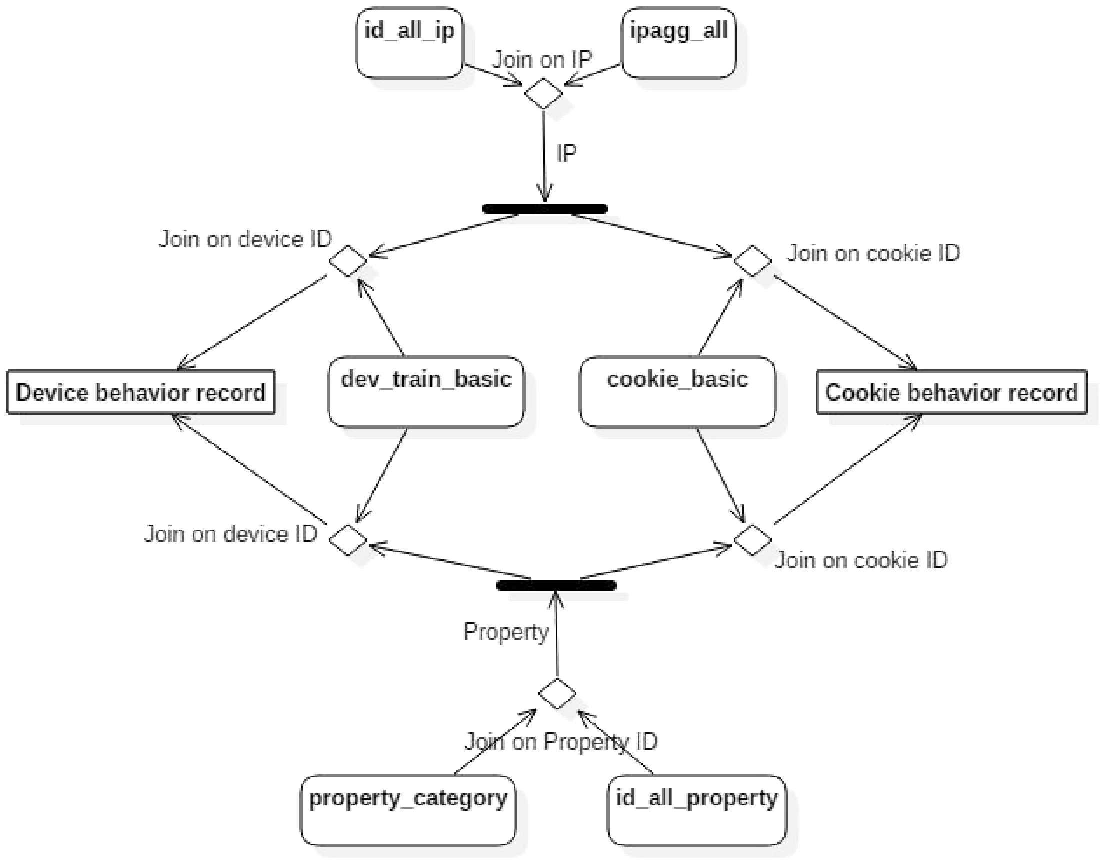

Figure 6.1: Schema join table

Joining files seems like a simple task until we look at the size of the main tables that store unique devices, cookies and IP addresses:

Name of the table	Amount of unique objects
device_train_basic.csv (devices)	142770
cookie_basic.csv (cookies)	2175520
ipagg_all.csv (IP addresses)	10097555

Table 6.1: Information on the main tables

As a result, we get approximately device-cookie pairs in the training set, which cannot be processed in an acceptable time.

Therefore, instead of looking over all possible pairs of objects, it was decided to find a list of cookie candidates for each device in advance and use only the selected pairs in training.

The filtering of candidates (downsampling) was performed under the following conditions:

I. The device and the cookie must have a common IP address that was accessed from both the mobile device and the computer.

II. As noted in [17] and [13], dynamic (cellular) IP addresses form a large number of pairs of cookies, most of which are negative examples, while static (non-cellular) addresses integrate most of the correct pairs. Therefore, it was decided to exclude links obtained through dynamic IP addresses from the training set.

III. All cookies with an undetermined owner (Drawbridge Handle equals to -1) were also excluded from candidates. The solution is dictated by the fact that such cookies can only be used as negative examples.

IV. The next filter based on the generalized information about the IP address. As suggested in [12], we are interested in IP addresses that are rarely used by users. Every cookie-device pair that both visited such an address also has high probability of having the same user.

   Considering this remark, we build an empirical rule that chose rarely visited addresses as candidates in the first place.

The results of each filter can be seen in Table 6.2 (the "Coverage percentage" column indicates the percentage of devices for which the candidate list contains at least one correct cookie). Filters were applied one after another.

Type of filtration	Amount of pairs (order of magnitude)	Coverage percentage (%)

Without filter		100
I		99,92
I+II		98,00
I+II+III		98.00
I+II+III+IV		97.09

Table 6.2: Results of filtration

As we can see, reducing the number of pairs by all filters leads to a decrease in coverage, for example, in the latter case, for 3% of all devices the desired cookie cannot be found in principle, but it was decided that this is a small fee for reducing the training sample in times.

The next step was to build a training set composed from the data of each pair of cookies. Based on the results of the experiments, the following fields were included in the final record of the set:

- all fields that aren't identifiers from the table device_train_basic.csv with basic information about devices (9 fields)
- all fields that aren't identifiers from the table cookie_basic.csv with basic information about cookies (9 fields)
- 56 new properties that summarize the behavior of the cookie-device pair on joint IP addresses (more details on new properties are described below)

A total of 74 properties were obtained.

The training set was constructed in such a way that if a cookie with a known owner (Drawbridge Handle <> -1) visited any IP address, then all other cookies of this owner were considered visiting this address, even if such data was not originally in the table. The ratio of positive and negative examples in the final set was approximately 1 : 3.

In order to characterize the new fields, we need to introduce some notation.

Denote by   the line of the table ipagg_all with the values of the fields (total_freq, c0, c1, c2). Similarly,   is the row of the id_all_ip table with all fields except the id and object type. The vector   is obtained by joining two tables by the IP address.

Denote   the set of all sites associated with the object   (device or cookie),   - the set of IP addresses of the object.

According to the algorithm described earlier, for each device d there was a candidate list   . Then for each pair   we defined:

- – a string describing the aggregate behavior of a pair on a common IP address
- – a set of all devices belonging to the same user as (excluding)
- a set of all cookies belonging to the same user as (excluding)

The table below briefly summarizes all the generated properties (the sum symbol means vector summation).

Property number	Property value
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	

28-47	
48-67	————
68-74	

Table 6.3: Generated properties

6.2 Models training and selection of parameters

Since we compared two approaches to the solution of the problem, we built two models. The actions described in this section and further were carried out independently on both ones. The main difference between the two approaches from the point of view of implementation was that for the ranking task the training and test set should have been grouped by the device id. In addition, the "objective" parameter for the binary classification problem was "binary: logistic", and for ranking - "rank: pairwise".

As described earlier, all the labeled data was divided into 2 halves: 85% of the devices went to the training set, the remaining 15% went to the test set and did not participate in the training.

In order to build the algorithm of gradient boosting on decision trees, we had to configure the following training parameters [19]:

- eta
- gamma
- max_depth
- subsample
- min_child_weight
- eval_metric

All of them were chosen so as to minimize the error on the training set.

To improve efficiency and generalize the properties, bagging was applied to the best model in the previous step. The cumulative probability was considered as the arithmetic mean of the predictions of all independent models. As in the case of learning parameters, their number was chosen so as to minimize the resulting error.

6.3 Postprocessing

Postprocessing was necessary for this task, because models themselves didn't get the output It was required to construct an algorithm by which the list of probabilities of all candidates could turn into the required list of cookies.

The simplest solution of all would be to take the cookie with the greatest probability and give it out as a response. However, this decision led to not the best results, and that's why.

The metric for the competition was named F0.5 score, which, as is known, gives preference to accuracy, i.e. the proportion of objects that are called positive by the classifier and in this case are actually positive. That means that the insufficient number of cookies selected is penalized more than an inadequate number of incorrect choices.

Given this feature, an algorithm was constructed that could select cookies based on their probability values. Here are the steps of the algorithm:

1. Sort all candidates by the probability value
2. The best candidate is added to the resulting list
3. For each other cookie , if

   then it is added to the resulting list, too.
4. To each cookie from the list all cookies of the same user are added

The parameter was chosen to minimize the error on the training set. It's important to note that we didn't have to train the model every time to find the optimal value of parameter – it was once trained and saved. After that we could use it for every experiment we wanted.

Moreover the algorithm has become more complicated due to the imposition of conditions. If number of already added in the final list cookies was small, become smaller, too, if number was large, then increased. This strategy allowed to slightly improve the final result.

In addition to creating the algorithm, at this stage we combine two ensembles of models into one. At the training stage, as it was done for bagging, it was impossible to do this because of different training samples. The combination was made as follows: composition of models for each device independently predicted its list of cookies, and the results were united.

7 Results

In this chapter we will perform key results that we got on every step and discuss them.

7.1 Preprocessing results

Filtering candidates according to the rules described in the relevant chapter proved to be quite effective (see Table 7.1) and allowed to significantly reduce the set size from non-trainable pairs to 740 thousands, which can be trained for an acceptable time. Unfortunately, along with downsampling, the coverage percentage (maximum accuracy that can be obtained on the set) also decreased, but not so much - only 3%.

Type of filtration	Number of pairs	Coverage percentage (%)
Without filtration		100
I		99,92
I+II		98,00
I+II+III		98.00
I+II+III+IV		97.09

Table 7.1: General results of filtration

It is noteworthy that the II filter excluded from the set all dynamic IP addresses that made up the majority of the ipagg_all table (see Table 1.4), which stores about 10 million addresses. Therefore we were provided by much more data than necessary for training.

Pairs obtained during preprocessing were divided into two subsets (see Table 7.2).

Name of set	Size
training set	634764
validation set	114046

Table 7.2: Division into subsets

7.2 Training results

Let’s compare the results obtained for each of the approaches.

	Binary classification problem	Learning-to-rank problem
Training parameters	eta: 0.1 gamma: 5 max_depth: 10 min_child_weight: 4 eval_metric: error objective: binary:logistic	eta: 0.2 gamma: 5 max_depth: 10 min_child_weight: 4 eval_metric: error objective: rank:pairwise
Number of models for bagging	6	6
Number of rounds of training	100	200
F0.5 score	0.86538	0.86588
F0.5 score with bagging	0.86546	0.86775

Table 7.3: Key results of the training stage

As you can see from Table 7.3, ranking required more rounds for training (i.e. time and resources), but it proved to be better than the first approach. In addition, bagging for ranking seemed to be more effective: its accuracy increased more than for the binary classification.

It is interesting to observe which of the parameters ultimately proved to be the most important for training [20]. The diagrams below show 10 and 20 most important properties according to the version of each model.

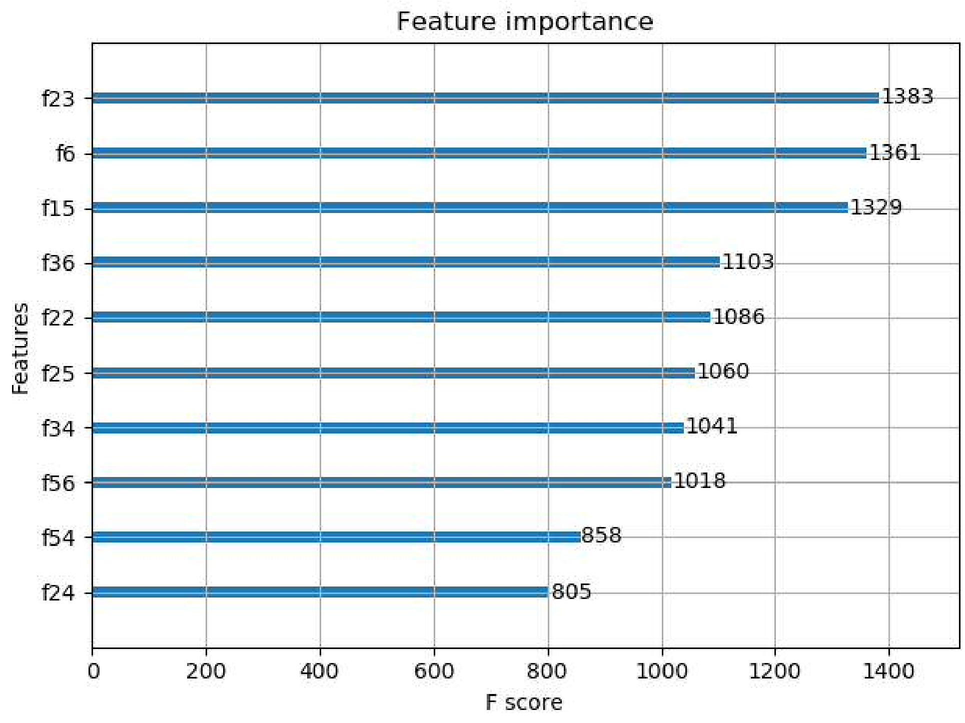

Figure 7.1: 10 most important properties according to binary classification

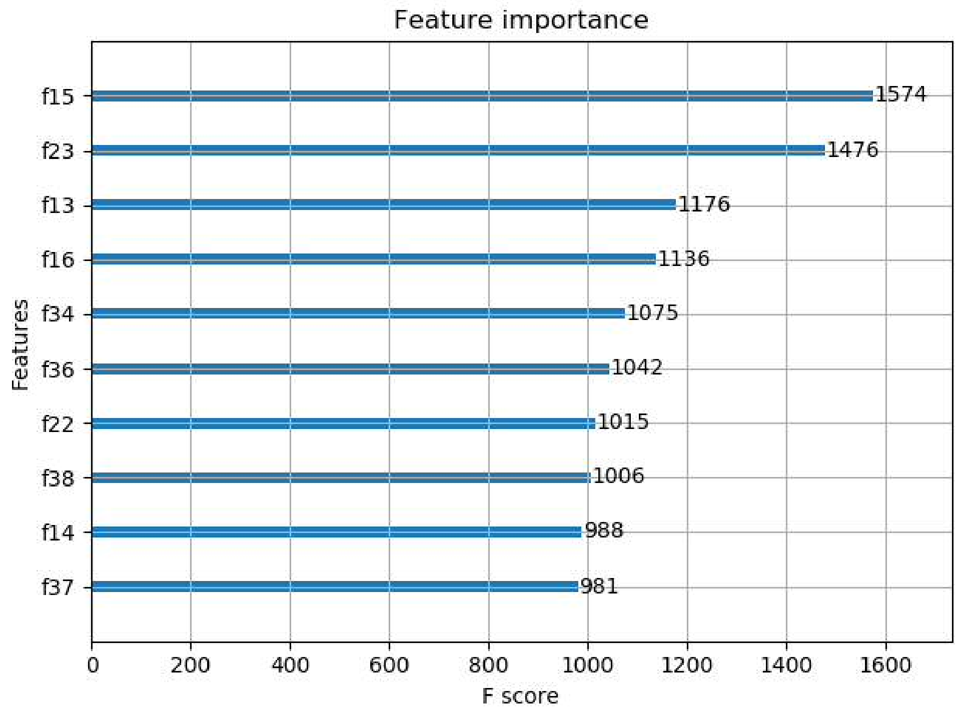

Figure 7.2: 10 most important properties according to ranking algorithm

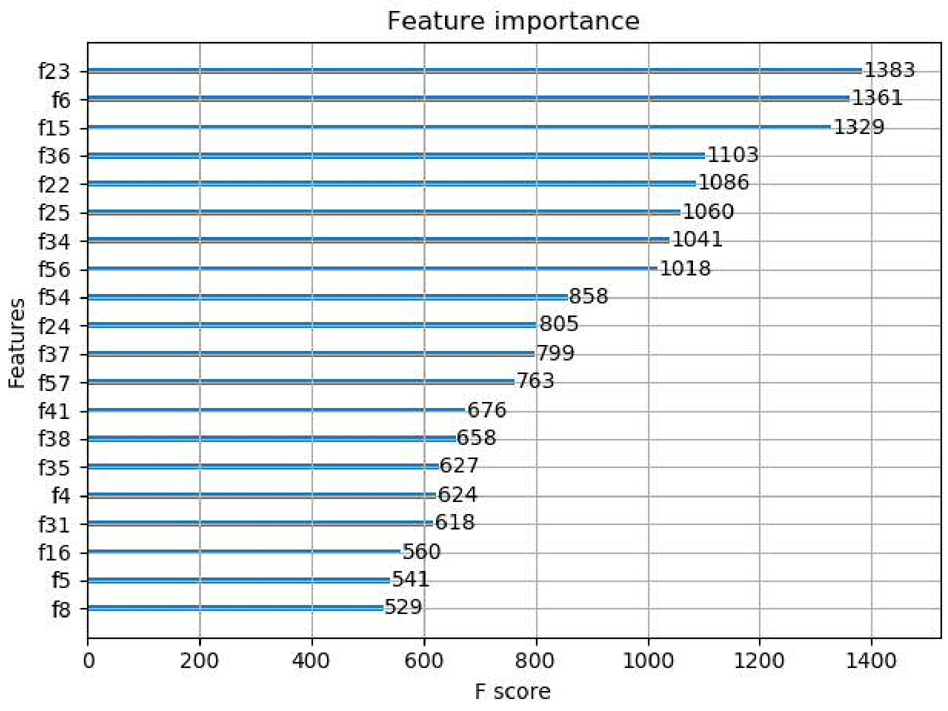

Figure 7.3: 20 most important properties according to binary classification

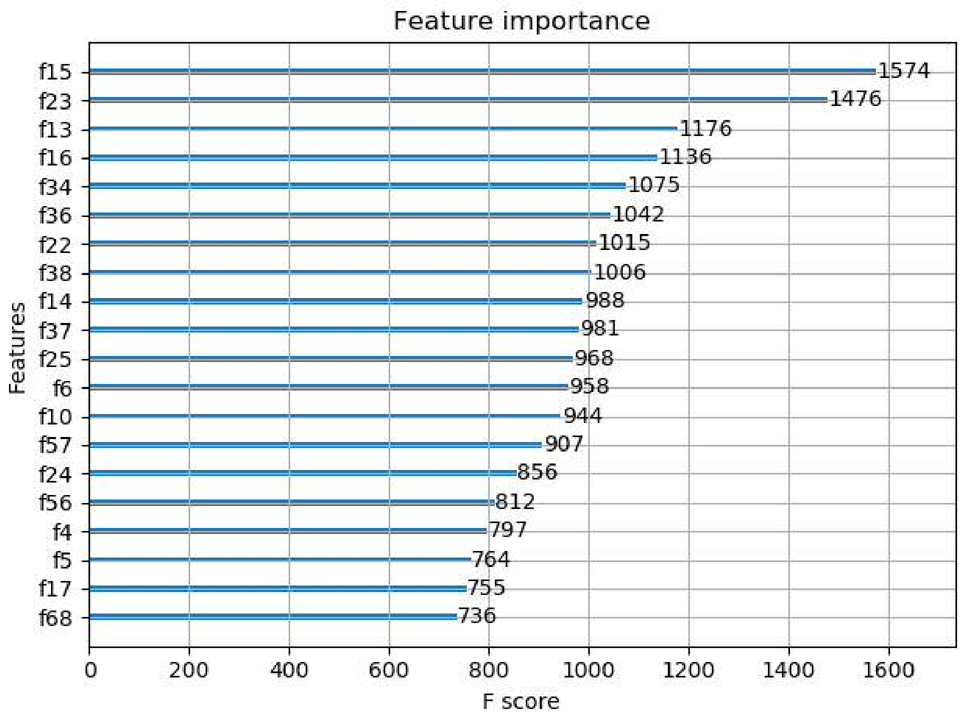

Figure 7.4: 20 most important properties according to ranking algorithm

Comparing the two models you can see that among the first ten important properties only 50% of them coincide, for the first twenty this value increases to 75%.

On average, both models recognize equally important the following properties:

- f23 – number of cookie's IP addresses
- f15 – cookie property Anonymous_5
- f36 –
- f22 – number of device's IP addresses
- f34 –

The binary classifier also actively uses:

- f6 – device property Anonymous_5
- f25 – number of other cookies of the same owner
- f56 – property f36, averaged over all IPs
- f54 – property f34, averaged over all IPs
- f24 – the number of mobile apps visited by the device

The ranking algorithm considers them necessary:

- f13 – cookie property Anonymous_c1
- f16 – cookie property Anonymous_6
- f38 –
- f14 – cookie property Anonymous_c2
- f37 –

As we see, the chosen properties largely depend on the approach that we use. The binary classification highlights general parameters of the cookie-device pair: among the unique properties there are both device and cookie fields. Ranking makes an explicit emphasis on the cookie: all its unique properties are somehow related to the behavior of the cookie, there is no one field describing the device. Moreover, f13, set by the ranking algorithm as the third most important feature, did not even hit the top twenty of the binary classification!

7.3 Postprocessing results

The following main results were obtained for postprocessing.

	Binary classification problem	Learning-to-rank problem
F0.5 score before postprocessing	0.86546	0.86775
Simple algorithm		
	0.96	0.84
F0.5 score	0.86736	0.86871
Advanced algorithm		
Parameters		
F0.5 score	0.86834	0.86941

Table 7.4: Key results of postprocessing

The ranking algorithm showed better results than the binary classification. As we see, the threshold in the ranking is lower - this means that the algorithm has a broader probability spread than the first approach, i.e. the most likely candidate is more distant from all the others. In general, for both algorithms postprocessing increased the initial estimate in approximately equal parts.

The combined model did not show any particular results: the complex selection algorithm gave F0.5 a measure of 0.8692, which is somewhere in between the results of both models. It was decided not to include it in the final table.

7.4 Summary

Table 7.5 stores the overall results of the work. Here is:

- I – gradient boosting model
- II – bagging
- III – simple algorithm for selecting cookies
- IV – advanced algorithm for selecting cookies

	Binary classification problem	Learning-to-rank problem
I	0.86538	0.86588
I+II	0.86546	0.86775
I+II+III	0.86736	0.86871
I+II+III+IV	0.86834	0.86941

Table 7.5: Summary results

The best result was shown by the ranking algorithm, to which was added bagging and advanced selection of cookies. The F0.5 score for this matter is equal to 0.86941 and it lies between the results of 4 and 5 places in the public leaderboard on Kaggle.

The binary classification in comparison with the ranking algorithm showed itself somewhat worse, but it required less resources and time for training.

In general, the difference in the results is not so great, but the approach significantly affects the optimal parameters and selection of the best properties: the ranking is sharpened to search for differences in cookies, therefore, focuses on the properties of the cookie and significantly differentiates the candidate list in probability. The binary classification pays more attention to the pair as a whole, looks not only at the parameters of the cookie, but also on the characteristics of the device. Differentiation for cookies is weak. To strengthen each model, you need to add more properties that it considers important: for binary classification - more information about joint behavior on IP addresses and sites/mobile applications, for ranking algorithm – more properties that distinguish cookies. This was demonstrated in the works of winners: for example, the team that took first place manually generated about 700 properties [17].

The cardinal difference of our work from all that participated in the competition in the size of training set: on average, for the leaders of the rating the number of pairs ranged from 4 [12] to 14 million [17], we were able to reduce the dimension to 700,000, which greatly accelerated the training time and simplified the work with data.

Based on the results of the work, it can be confidently asserted that both approaches to the problem (as to binary classification and as to learning-to-rank problem) are justified: the algorithms showed good results of the same order. But it should be noted that for effective implementation of each of the approaches it is necessary to build their training set so as to take into account their features.

Conclusion

In the paper we solved the problem of cross-device connection between computers and mobile devices. We had to find those which belong to the same user based on the behavior of objects and the IP addresses they used. To solve the problem various machine learning techniques and algorithms were applied.

We considered two basic approaches to the solution: as to the task of binary classification or ranking. For each interpretation a rigorous formulation was defined and the methods of solution chosen. Gradient boosting on decision trees became the main learning algorithm for each approach. We used Python 3.6.0 for all processing, training and data analysis procedures. We found XGBoost framework extremely efficient in solving the problem.

Extensive preliminary work with the data was carried out. This work included creating a training set and generating properties based on the data presented in the tables. We had to add a number of filters related to the properties of IP addresses and cookies in order to reduce initial size of candidates' set. In the course of the work, it was shown that a large amount of data is not required for a successful solution of the problem. A detailed description of the objects and their behavior on static IP addresses is enough for 97% coverage. Set that was built according to this rules took up only a little space, which became an important indicator for training.

Selection model parameters and postprocessing stages revealed that although the two approaches showed similar results, they reach those across fundamentally different means: ranking model focuses more on cookies than the device, the binary classification seems to be more balanced. At the same time, both models identify the number of IP addresses of cookies and devices as one of the most important parameters. The first approach required less time for training than the second. Adding bagging and postprocessing improved the results of both models.

Ranking algorithm showed itself better than binary classification, it has a final F0.5 score on the test set equal to 0.86941. Behind it, with a small margin, followed a binary classification with a score of 0.86834. The obtained values lie between the results of 4 and 5 places in the public leaderboard on Kaggle.

Further steps to improve the algorithm can be an extension of the list of generated properties, which must be selected according to the features of the approach (binary classification or ranking). You can also try to combine the models of the two approaches at earlier stages (learning, not postprocessing).

Based on the results of the work, it can be said with certainty that cross-device marketing is closer than ever: modern algorithms allow to link devices and computers belonging to one person with high accuracy and do not need to collect large amounts of information. The choice of the approach was not as critical as it might seem - to get good results it is enough to clearly represent the essence of the approach and accurately generate the training sample according to its essence.

Bibliography

[1] "Google Analytics remarketing lists go cross-device May 15," [Online]. Available: https://marketingland.com/google-analytics-remarketing-lists-go-cross-device-may-15-211248. [Accessed 2018].

[2] "Lotame," [Online]. Available: https://www.lotame.com/.

[3] "Tapad," [Online]. Available: http://www.tapad.com/.

[4] "Drawbridge," [Online]. Available: https://drawbridge.com/.

[5] "ICDM 2015: Drawbridge Cross-Device Connections," [Online]. Available: https://www.kaggle.com/c/icdm-2015-drawbridge-cross-device-connections#description.

[6] A. Fonarev, "Overview of Boosting Methods," Moscow, 2012.

[7] R. L. R. L. Hyafil, "Constructing optimal binary decision trees is NP-complete," Information processing letters, 1976.

[8] Open Data Science, "Open course of machine learning. Topic 5. Compositions: bagging, Random Forest," [Online]. Available: https://habr.com/company/ods/blog/324402/.

[9] D. Bazhenov, "Classifier estimation (recall, precision, F-score)," [Online]. Available: http://bazhenov.me/blog/2012/07/21/classification-performance-evaluation.html.

[10] Y. Jin, "Tree Boosting With XGBoost," [Online]. Available: https://medium.com/syncedreview/tree-boosting-with-xgboost-why-does-xgboost-win-every-machine-learning-competition-ca8034c0b283.

[11] Open Data Science, "Open course of machine learning. Topic 10. Gradient boosting," [Online]. Available: https://habr.com/company/ods/blog/327250/.

[12] R. Díaz-Morales, "Cross-Device Tracking: Matching Devices and Cookies," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, Atlantic City, 2015.

[13] S. R. S. a. R. C. M. Landry, "Multi-layer Classification: ICDM 2015 Drawbridge Cross-Device Connections Competition," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, Atlantic City, 2015.

[14] J. L. X. W. a. W. Y. M. S. Kim, "Connecting Devices to Cookies via Filtering, Feature Engineering, and Boosting," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, Atlantic City, 2015.

[15] T. R. A. a. O. Renov, "Machine Learning Approach to Identify Users Across Their Digital Devices," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015.

[16] G. K. a. C. Rong, "Cross-Device Consumer Identification," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, Atlantic City, 2015.

[17] J. Walthers, "Learning to Rank for Cross-Device Identification," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, Atlantic City, 2015.

[18] "XGBoost official repository," [Online]. Available: https://github.com/dmlc/xgboost. [Accessed 2018].

[19] A. Dyakonov, "Gradient boosting," [Online]. Available: https://alexanderdyakonov.files.wordpress.com/2017/06/book_boosting_pdf.pdf.

[20] J. Brownlee, "Feature Importance and Feature Selection With XGBoost in Python," [Online]. Available: https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/.

[21] "MachineLearning.ru," [Online]. Available: http://www.machinelearning.ru/wiki/.

[22] Open Data Science, "Metrics in machine learning problems," [Online]. Available: https://habr.com/company/ods/blog/328372/.

[23] K. Voroncov, "Learning to rank," [Online]. Available: http://www.machinelearning.ru/wiki/images/8/89/Voron-ML-Ranking-slides.pdf.

[24] J. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *Reitz Lecture,* 1999.

Contents of enclosed CD

src	the directory of source codes
text	the thesis text directory
thesis.pdf	the thesis text in PDF format

Appendix

Below you can see the main code files:

1. DataBuilder

```
from Variables import *
from CookieLibrary import *
from Analytics import *
from TrainLibrary import calculateF05

validationPercent = 0.15

def createDatasets(dictHandle, dictDevice, dictCookie, devicesTrain,
cookies, labels, groups, whosDevice, time1):

    print('STEP: Loading Properties File')
    devProperties = initPropDict(propFile, dictDevice, dictCookie)
    (time1) = measureTime(time1)

    print('STEP: Loading IP Files')
    XIPS = loadIPAGG(ipaggFile)
    (time1) = measureTime(time1)

    print('STEP: Loading IPs')
    (IPDev, IPCoo, deviceIPs, cookieIPs) = loadIPS(ipFile,
dictDevice, dictCookie, XIPS, groups)
    (time1) = measureTime(time1)

    print('STEP: Initial selection of candidates')
    deviceSize = len(dictDevice)
    validationSize = np.int(deviceSize * validationPercent)
    print("Validation size: ", validationSize)

    (candidatesTrain, totalSizeTrain) =
selectCandidates(devicesTrain[0:-validationSize, :], cookies, IPCoo,
IPDev, deviceIPs, dictHandle)
    (candidatesTest, totalSizeTest) =
selectCandidates(devicesTrain[-validationSize:-1, :], cookies,
IPCoo, IPDev, deviceIPs, dictHandle)

    (time1) = measureTime(time1)
    # all available devices are divided int 2 parts:
    # 85% - training set
    # 15% - validation set
    # training set divided once more to train anf test set
    print('STEP: Creating training dataset')
    (xTrain, originalIndexTrain) = createDataSet(candidatesTrain,
devicesTrain, cookies, deviceIPs, cookieIPs, groups, whosDevice,
devProperties)
    time1 = measureTime(time1)

    print('STEP: Creating test dataset')
    (xTest, originalIndexTest) = createDataSet(candidatesTest,
devicesTrain, cookies, deviceIPs, cookieIPs, groups, whosDevice,
devProperties)

    print("Train set size:", xTrain.shape[0])
    print("Test set size:", xTest.shape[0])
    time1 = measureTime(time1)

    print('STEP: Creating labels')
    yTrain = createTrainingLabels(candidatesTrain, labels)
    time1 = measureTime(time1)

    print('STEP: Saving training and test data')
    saveDatasets(xTrain, yTrain, xTest, originalIndexTrain,
originalIndexTest, groups, labels, genDataPath)
    time1 = measureTime(time1)

    return xTrain, yTrain, xTest, originalIndexTrain,
originalIndexTest, time1


def calcSizeAndCoverage(dictHandle, dictDevice, dictCookie,
devicesTrain, cookies, labels, groups, whosDevice, time1):

    print('STEP: Loading IP Files')
    XIPS = loadIPAGG(ipaggFile)
    (time1) = measureTime(time1)

    print('STEP: Loading IPs')
    (IPDev, IPCoo, deviceIPs, cookieIPs) = loadIPS(ipFile,
dictDevice, dictCookie, XIPS, groups)
    (time1) = measureTime(time1)

    print('STEP: Selecting candidates')
    (candidates, totalSize) = selectCandidates(devicesTrain,
cookies, IPCoo, IPDev, deviceIPs, dictHandle)
    (time1) = measureTime(time1)

    print('STEP: Calculating coverage')
    coverage = calcCoverage(labels, candidates)
    (time1) = measureTime(time1)

    print('STEP: Calculating max F05')
    trueLabels = findTrueLabels(labels, candidates)
    F05 = calculateF05(trueLabels, labels)
    (time1) = measureTime(time1)
    print("Total size: ", totalSize)
    print("Coverage: ", coverage)
    print("Max F05: ", F05)

    return totalSize, coverage


def findTrueLabels(labels, candidates):
    trueLabels = dict()
    for deviceId, cookies in labels.items():
        localCandidates = candidates.get(deviceId, set())
        trueLabels[deviceId] = set()
        for cookie in cookies:
            if cookie in localCandidates:
                trueLabels[deviceId].add(cookie)

    return trueLabels
```

2. CookieLibrary

```
import csv
import numpy as np
import re
from collections import defaultdict
import time
import pickle

def initDictionaries(trainFile, cookieFile):
    deviceList = list()
    cookieList = list()
    handleList = list()
    devTypeList = list()
    devOsList = list()
    computerOsList = list()
    computerVList = list()
    countryList = list()
    annC1List = list()
    annC2List = list()

    with open(trainFile, 'rt') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',')
        spamreader.__next__()
        for row in spamreader:
            handleList.append(row[0])
            deviceList.append(row[1])
            devTypeList.append(row[2])
            devOsList.append(row[3])
            countryList.append(row[4])
            annC1List.append(row[6])
            annC2List.append(row[7])
    deviceList = list(set(deviceList))
    handleList = list(set(handleList))
    devTypeList = list(set(devTypeList))
    devOsList = list(set(devOsList))
    countryList = list(set(countryList))
    annC1List = list(set(annC1List))
    annC2List = list(set(annC2List))

    devNum = len(deviceList)
    print("Train device number: %s" % devNum)

    with open(cookieFile, 'rt') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',')
        spamreader.__next__()
        for row in spamreader:
            handleList.append(row[0])
            cookieList.append(row[1])
            computerOsList.append(row[2])
            computerVList.append(row[3])
            countryList.append(row[4])
            annC1List.append(row[6])
            annC2List.append(row[7])

    cookieList = list(set(cookieList))
    handleList = list(set(handleList))
    computerOsList = list(set(computerOsList))
    computerVList = list(set(computerVList))
    countryList = list(set(countryList))
    annC1List = list(set(annC1List))
    annC2List = list(set(annC2List))

    print("Cookie number: %s" % len(cookieList))

    dictHandle = list2Dict(handleList)
    dictDevice = list2Dict(deviceList)
    dictCookie = list2Dict(cookieList)
    dictDevType = list2Dict(devTypeList)
    dictDevOs = list2Dict(devOsList)
    dictComputerOs = list2Dict(computerOsList)
    dictComputerV = list2Dict(computerVList)
    dictCountry = list2Dict(countryList)
    dictAnnC1 = list2Dict(annC1List)
    dictAnnC2 = list2Dict(annC2List)

    return (dictDevice, dictCookie, dictHandle, dictDevType,
dictDevOs, dictComputerOs, dictComputerV, dictCountry,
            dictAnnC1, dictAnnC2)


def list2Dict(list):
    newDict = dict()
    for i in range(len(list)):
        newDict[list[i]] = i
    return newDict


def loadDevices(trainFile, dictHandle, dictDevice, dictDevType,
dictDevOs, dictCountry, dictAnnC1, dictAnnC2):
    with open(trainFile, 'rt') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',')
        spamreader.__next__()
        rowNumber = sum(1 for row in spamreader)

    XDevices = np.zeros((rowNumber, 11))

    rowNumber = 0
    with open(trainFile, 'rt') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',')
        spamreader.__next__()
        for row in spamreader:
            XDevices[rowNumber, 0] = dictHandle[row[0]]
            XDevices[rowNumber, 1] = dictDevice[row[1]]
            XDevices[rowNumber, 2] = dictDevType[row[2]]
            XDevices[rowNumber, 3] = dictDevOs[row[3]]
            XDevices[rowNumber, 4] = dictCountry[row[4]]
            XDevices[rowNumber, 5] = np.float_(row[5])
            XDevices[rowNumber, 6] = dictAnnC1[row[6]]
            XDevices[rowNumber, 7] = dictAnnC2[row[7]]
            XDevices[rowNumber, 8] = np.float_(row[8])
            XDevices[rowNumber, 9] = np.float_(row[9])
            XDevices[rowNumber, 10] = np.float_(row[10])
            rowNumber += 1

    return XDevices


def loadCookies(cookieFile, dictHandle, dictCookie, dictComputerOs,
dictComputerV, dictCountry, dictAnnC1, dictAnnC2):
    maxindex = max(dictCookie.values())
    XCookies = np.zeros((maxindex + 1, 11))

    with open(cookieFile, 'rt') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',')
        spamreader.__next__()
        for row in spamreader:
            cookieId = np.int(dictCookie[row[1]])
            XCookies[cookieId, 0] = dictHandle[row[0]]
            XCookies[cookieId, 1] = dictCookie[row[1]]
            XCookies[cookieId, 2] = dictComputerOs[row[2]]
            XCookies[cookieId, 3] = dictComputerV[row[3]]
            XCookies[cookieId, 4] = dictCountry[row[4]]
            XCookies[cookieId, 5] = np.float_(row[5])
            XCookies[cookieId, 6] = dictAnnC1[row[6]]
            XCookies[cookieId, 7] = dictAnnC2[row[7]]
            XCookies[cookieId, 8] = np.float_(row[8])
            XCookies[cookieId, 9] = np.float_(row[9])
            XCookies[cookieId, 10] = np.float_(row[10])

    return XCookies


def initPropDict(fileProps, dictDevice, dictCookie):
    devProps = dict()

    with open(fileProps) as fp:
        fp.readline()

        for line in fp:
            matchObj = re.match(r'([a-zA-Z0-9_]*),([0-1]),{(([(a-zA-
Z0-9.(),\-_]*)}', line, flags=0)

            if matchObj.group(2) == '0':
                props = re.findall(r'\((.*?)\)', matchObj.group(3))
                valProps = dict()
                for prop in props:
                    propV = prop.split(',')
                    valProps[propV[0]] = np.float_(propV[1])
                deviceId = dictDevice.get(matchObj.group(1), -1)
                if deviceId > -1:
                    devProps[deviceId] = valProps

    return devProps


def creatingLabels(devices, cookies, dictHandle):

    handleData = dict()
    unknown = dictHandle['-1']
    handles = np.unique(cookies[:, 0])
    for i in range(len(handles)):
        if handles[i] != unknown:
            handleData[handles[i]] = dict()
            handleData[handles[i]]['Devices'] = set()
            handleData[handles[i]]['Cookies'] = set()

    (nDevices, nDim) = devices.shape

    for i in range(nDevices):
        handleData[devices[i, 0]]['Devices'].add(devices[i, 1])

    (nCookies, nDim) = cookies.shape

    for i in range(nCookies):
        if cookies[i, 0] != unknown:
            handle = handleData.get(cookies[i, 0])
            handle['Cookies'].add(cookies[i, 1])
    labels = dict()
    groups = dict()
    whosDevice = dict()

    for id, handle in handleData.items():
        for dev in handle['Devices']:
            labels[dev] = handle['Cookies']
        for cookie in handle['Cookies']:
            groups[cookie] = handle['Cookies']
            whosDevice[cookie] = handle['Devices']

    for i in range(nCookies):
        if cookies[i, 0] == unknown:
            name = cookies[i, 1]
            cookieSet = set()
            cookieSet.add(name)
            groups[name] = cookieSet  # каждый в своем мирке

    return labels, groups, whosDevice
    #return labels


def loadIPAGG(ipaggFile):
    XIPS = dict()

    with open(ipaggFile, 'rt') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=',')
        spamreader.__next__()

        inx = 0
        for row in spamreader:

            if inx % 500000 == 0:
                print(inx)

            if row[2] == '0':
                data = np.zeros(4)
                # data[0] = np.int32(row[2])
                data[0] = np.int32(row[3])
                data[1] = np.int32(row[4])
                data[2] = np.int32(row[5])
                data[3] = np.int32(row[6])
                XIPS[row[1]] = data

            inx += 1

    print("Total amount of IP is %s" % len(XIPS))
    return XIPS


def loadIPS(ipFile, dictDevice, dictCookie, IPs, groups):
    deviceIPs = dict()
    cookieIPs = dict()
    IPDev = defaultdict(set)
    IPCoo = defaultdict(set)

    with open(ipFile) as fp:
        fp.readline()

        ind = 0
        for line in fp:
            matchObj = re.match(r'([a-zA-Z0-9_]*),([0-1]),{(([(a-zA-
Z0-9(),\-_]*)}', line, flags=0)
            ips = re.findall(r'(\w*,\w*,\w*,\w*,\w*,\w*,\w*)',
matchObj.group(3))

            if ind % 100000 == 0:
                print(ind)

            ipRecord = dict()
            for ip in ips:
                ipInfo = ip.split(',')
                ipGenInfo = IPs.get(ipInfo[0], np.empty(0))

                if len(ipGenInfo) > 0:
                    arr = np.zeros(10)
                    arr[0] = np.int_(ipInfo[1])
                    arr[1] = np.int_(ipInfo[2])
                    arr[2] = np.int_(ipInfo[3])
                    arr[3] = np.int_(ipInfo[4])
                    arr[4] = np.int_(ipInfo[5])
                    arr[5] = np.int_(ipInfo[6])

                    arr[6] = np.int_(ipGenInfo[0])
                    arr[7] = np.int_(ipGenInfo[1])
                    arr[8] = np.int_(ipGenInfo[2])
                    arr[9] = np.int_(ipGenInfo[3])

                    ipRecord[ipInfo[0]] = arr

            if matchObj.group(2) == '0':
                deviceId = dictDevice.get(matchObj.group(1), -1)
                if deviceId > -1:
                    deviceIPs[deviceId] = ipRecord
                    for k in ipRecord.keys():
                        IPDev[k].add(deviceId)
            else:
                cookieId = dictCookie[matchObj.group(1)]
                cookieIPs[cookieId] = ipRecord
                for ip in ipRecord.keys():
                    IPCoo[ip].add(cookieId)

            ind += 1
    for k, v in groups.items():
        if len(v) > 1:
            for cook1 in v:
                for cook2 in v:
                    if cook1 != cook2:
                        d1 = cookieIPs[cook1]
                        d2 = cookieIPs[cook2]
                        for n1, n2 in d1.items():
                            if n1 not in d2.keys():
                                d2[n1] = n2
                                IPCoo[n1].add(cook2)

    return IPDev, IPCoo, deviceIPs, cookieIPs


def measureTime(time1):
    time2 = time1
    time1 = time.time()
    print("--- %s seconds ---" % round(time1 - time2, 2))
    return time1


def selectCandidates(devices, cookies, IPCoo, IPDev, deviceIPS,
dictHandle):

    deviceIds = np.unique(devices[:, 1])
    candidates = dict()
    unknown = dictHandle['-1']
    totalSize = 0
    index = 0
    for deviceId in deviceIds:
        candidateSet = set()

        if index % 10000 == 0:
            print(index)

        ips = deviceIPS[deviceId]
        for ip in ips.keys():
            if len(IPDev.get(ip, set())) <= 10 and len(IPCoo.get(ip,
set())) <= 20:
                localCandidates = IPCoo[ip]
                for candidate in localCandidates:
                    if cookies[np.int(candidate), 0] != unknown:
                        candidateSet.add(candidate)

        if len(candidateSet) == 0:
            for ip in ips:
                if len(IPDev.get(ip, set())) <= 25 and
len(IPCoo.get(ip, set())) <= 50:
                    localCandidates = IPCoo[ip]
                    for candidate in localCandidates:
                        if cookies[np.int(candidate), 0] != unknown:
                            candidateSet.add(candidate)

        if len(candidateSet) == 0:
            for ip in ips:
                localCandidates = IPCoo[ip]
                for candidate in localCandidates:
                    if cookies[np.int(candidate), 0] != unknown:
                        candidateSet.add(candidate)

        if len(candidateSet) == 0:
            for ip in ips:
                localCandidates = IPCoo[ip]
                for candidate in localCandidates:
                    candidateSet.add(candidate)

        totalSize += len(candidateSet)
        candidates[deviceId] = candidateSet
        index += 1

    print("Pairs total: %s" % totalSize)
    return candidates, totalSize


def createDataSet(candidates, devices, cookies, deviceIPS,
cookieIPS, groups, whosDevice, devProps):

    originalIndex = dict()
    numPatterns = 0
    for deviceId, cookieIds in candidates.items():
        numPatterns = numPatterns + len(cookieIds)

    added = 0
    index = 0
    for deviceId, cookieIds in candidates.items():

        if index % 5000 == 0:
            print(index)

        deviceRecord = devices[devices[:, 1] == deviceId,
np.array([2, 3, 4, 5, 6, 7, 8, 9, 10])]

        indivIndex = dict()
        deviceSet = set()
        deviceSet.add(deviceId)
        deviceIps = set(deviceIPS.get(deviceId, dict()).keys())
        deviceProps = set(devProps.get(deviceId, dict()).keys())

        for cookieId in cookieIds:
            cookieRecord = cookies[np.int(cookieId), np.array([2, 3,
4, 5, 6, 7, 8, 9, 10])]

            row = np.concatenate((deviceRecord, cookieRecord))
            cookieIps = set(cookieIPS.get(cookieId, dict()).keys())
            ips = (deviceIps & cookieIps)
            otherDevices = set(whosDevice.get(cookieId, set())) -
deviceSet

            devp = set()
            devi = set()
            for odev in otherDevices:
                devp = devp | set(devProps.get(odev, dict()).keys())
                devi = devi | set(deviceIPS.get(odev,
dict()).keys())

            intersec = np.float_(len(devp & deviceProps))
            interseci = np.float_(len(devi & deviceIps))

            if intersec > 0:
                intersec = intersec / np.float_(len(deviceProps))

            if interseci > 0:
                intersec = intersec / np.float_(len(deviceIps))

            row = np.concatenate((row,
np.array([np.float_(len(otherDevices))])))
            row = np.concatenate((row,
np.array([np.float_(intersec)])))

            row = np.concatenate((row,
np.array([np.float_(interseci)])))

            row = np.concatenate((row,
np.array([np.float_(len(ips))])))
            row = np.concatenate((row,
np.array([np.float_(len(deviceIps))])))
            row = np.concatenate((row,
np.array([np.float_(len(cookieIps))])))

            row = np.concatenate((row,
np.array([np.float_(len(deviceProps))])))

            row = np.concatenate((row,
np.array([np.float_(len(groups.get(cookieId, set())))])))
            row = np.concatenate((row,
np.array([np.float_(len(groups.get(cookieId, set()) &
cookieIds))])))
            row = np.concatenate((row,
np.array([np.float_(len(ips))])))

            iprow = np.zeros(20)
            niprows = 0
            for ip in ips:
                iprow = iprow +
np.concatenate((deviceIPS[deviceId][ip].reshape(-1),
cookieIPS[cookieId][ip].reshape(-1)))
                niprows = niprows + 1

            if niprows > 0:
                meaniprows = iprow / np.float_(niprows)
            else:
                meaniprows = iprow

            row = np.concatenate((row.reshape(-1), iprow.reshape(-
1)))
            row = np.concatenate((row.reshape(-1),
meaniprows.reshape(-1)))
            row = np.concatenate((row.reshape(-1), (iprow[0:6] -
iprow[10:16]).reshape(-1)))

            if added == 0:
                xTrain = np.zeros((numPatterns, len(row)))

            indivIndex[cookieId] = added

            xTrain[added, :] = row

            added = added + 1
        originalIndex[deviceId] = indivIndex

        index += 1

    print("xTrain shape is {}".format(xTrain.shape))
    return xTrain, originalIndex


def createTrainingLabels(candidates, labels):
    numPatterns = 0

    for k, v in candidates.items():
        numPatterns = numPatterns + len(v)

    yTrain = np.zeros(numPatterns)

    added = 0
    for k, v in candidates.items():
        for coo in v:
            if coo in labels[k]:
                yTrain[added] = 1.0
            added = added + 1

    return yTrain
```

```
def saveDatasets(xTrain, yTrain, xTest, originalIndexTrain,
originalIndexTest, groups, labels, genDataPath):
    np.save(genDataPath + 'xTrain.npy', xTrain)
    np.save(genDataPath + 'xTest.npy', xTest)
    np.save(genDataPath + 'yTrain.npy', yTrain)

    modelFile = genDataPath + 'originalIndexTrain.pkl'
    f = open(modelFile, "wb")
    pickle.dump(originalIndexTrain, f)
    f.close()

    modelFile = genDataPath + 'originalIndexTest.pkl'
    f = open(modelFile, "wb")
    pickle.dump(originalIndexTest, f)
    f.close()

    modelFile = genDataPath + 'groups.pkl'
    f = open(modelFile, "wb")
    pickle.dump(groups, f)
    f.close()

    modelFile = genDataPath + 'labels.pkl'
    f = open(modelFile, "wb")
    pickle.dump(labels, f)
    f.close()
```

3. ModelBuilder

```
from TrainLibrary import *
from Variables import *
from CookieLibrary import *

os.environ['PATH'] = os.environ['PATH'] + ';' + pathXGBoost

def buildModel(time1):

    print('STEP: Loading datasets')
    (xTrain, yTrain, xTest, originalIndexTrain, originalIndexTest,
groups, labels) = loadDatasets(genDataPath)
    (time1) = measureTime(time1)

    print('STEP: Training Supervised Learning')
    (resultadosVal, resultadosTest, classifiers) \
        = fullTraining(xTrain, yTrain, xTest, originalIndexTrain,
groups)
    (time1) = measureTime(time1)

    print('STEP: Post Processing')
    (validate, thTR) = bestSelection(resultadosVal,
originalIndexTrain, np.array([1.0, 0.9]), groups, 1)
    (test, thTest) = bestSelection(resultadosTest,
originalIndexTest, np.array([1.0, 0.9]), groups, 1)
```

```
    (time1) = measureTime(time1)

    F05 = calculateF05(validate, labels)
    print("F05 validation ", F05)
    (time1) = measureTime(time1)

    F05 = calculateF05(test, labels)
    print("F05 test ", F05)
    (time1) = measureTime(time1)

    print('STEP: Saving model')
    saveModel(modelPath, classifiers)
    (time1) = measureTime(time1)

    return classifiers, time1


def buildPairwiseModel(time1):
    print('STEP: Loading datasets')
    (xTrain, yTrain, xTest, originalIndexTrain, originalIndexTest,
groups, labels) = loadDatasets(genDataPath)
    (time1) = measureTime(time1)

    print('STEP: Prepare data')
    (orderedXTrain,     orderedIndexTrain,     orderedYTrain)     =
orderByDevice(xTrain, originalIndexTrain, yTrain)
    (orderedXTest,    orderedIndexTest)    =    orderByDevice(xTest,
originalIndexTest)
    (time1) = measureTime(time1)

    print('STEP: Training Supervised Learning')
    (resultadosVal, resultadosTest, classifiers) \
        =     fullTrainingPairwise(orderedXTrain,     orderedYTrain,
orderedXTest, orderedIndexTrain, groups)
    (time1) = measureTime(time1)


    print('STEP: Post Processing')
    (validate,       thTR)       =       bestSelection(resultadosVal,
originalIndexTrain, np.array([1.0, 0.9]), groups, 1)
    (test, thTest) = bestSelection(resultadosTest, originalIndexTest,
np.array([1.0, 0.9]), groups, 1)
    (time1) = measureTime(time1)

    F05 = calculateF05(validate, labels)
    print("F05 validation ", F05)

    F05 = calculateF05(test, labels)
    print("F05 test ", F05)
    (time1) = measureTime(time1)

    print('STEP: Saving model')
    saveModel(pairwiseModelPath, classifiers)
    (time1) = measureTime(time1)

    return classifiers, time1


def postProcessingEvaluation(time1):

    print('STEP: Loading datasets')
    (xTrain, yTrain, xTest, originalIndexTrain, originalIndexTest,
groups, labels) = loadDatasets(genDataPath)
    (time1) = measureTime(time1)

    print('STEP: Loading model')
    classifiers = loadModel(modelPath)
    (time1) = measureTime(time1)

    print('STEP: Post Processing')
    resultsTrain = np.zeros(xTrain.shape[0])
    resultsTest = np.zeros(xTest.shape[0])
    index = 0
    for classifier in classifiers:
        index += 1
        print(index)

        pTrain = predictXGBoost(xTrain, classifier)
        resultsTrain = resultsTrain + pTrain
        pTest = predictXGBoost(xTest, classifier)
        resultsTest = resultsTest + pTest

    resultsTrain = resultsTrain / np.float_(len(classifiers))
    resultsTest = resultsTest / np.float_(len(classifiers))

    betas = [0.8+x*0.02 for x in range(1)]
    print(betas)
    for beta in betas:
        print("----> Beta is", beta)

        (resultTrain, thresholdTrain) = bestSelection(resultsTrain,
originalIndexTrain, np.array([1.0, 0.96]), groups, beta)
        F05 = calculateF05(resultTrain, labels)
        print("Train F0.5", F05)

        (resultTest, thresholdTest) = bestSelection(resultsTest,
originalIndexTest, np.array([1.0, 0.96]), groups, beta)
        F05 = calculateF05(resultTest, labels)
        print("Test F0.5", F05)
    (time1) = measureTime(time1)


def postProcessingPairwise(time1):
    print('STEP: Loading datasets')
    (xTrain,  yTrain,  xTest,  originalIndexTrain,  originalIndexTest,
groups, labels) = loadDatasets(genDataPath)
    (time1) = measureTime(time1)

    print('STEP: Prepare data')
    (orderedXTrain,     orderedIndexTrain,     orderedYTrain)     =
orderByDevice(xTrain, originalIndexTrain, yTrain)
    (orderedXTest,    orderedIndexTest)    =    orderByDevice(xTest,
originalIndexTest)

    print('STEP: Loading model')
    classifiers = loadModel(pairwiseModelPath)
    (time1) = measureTime(time1)

    print('STEP: Post Processing')
    resultsTrain = np.zeros(orderedXTrain.shape[0])
    resultsTest = np.zeros(orderedXTest.shape[0])
    index = 0
    for classifier in classifiers:
        index += 1
        print(index)

        pTrain = predictXGBoost(orderedXTrain, classifier)
        resultsTrain = resultsTrain + pTrain
        pTest = predictXGBoost(orderedXTest, classifier)
        resultsTest = resultsTest + pTest

    resultsTrain = resultsTrain / np.float_(len(classifiers))
    resultsTest = resultsTest / np.float_(len(classifiers))

    betas = [0.8 + x * 0.02 for x in range(1)]
    print(betas)
    for beta in betas:
        print("Beta is", beta)

        (resultTrain,  thresholdTrain)  =  bestSelection(resultsTrain,
orderedIndexTrain, np.array([1.0, 0.84]), groups, beta)
        F05 = calculateF05(resultTrain, labels)
        print("Train F0.5", F05)

        (resultTest,   thresholdTest)   =   bestSelection(resultsTest,
orderedIndexTest, np.array([1.0, 0.84]), groups, beta)
        F05 = calculateF05(resultTest, labels)
        print("Test F0.5", F05)

    (time1) = measureTime(time1)


def calcRatio(time1):
    print('STEP: Loading datasets')
    (xTrain,  yTrain,  xTest,  originalIndexTrain,  originalIndexTest,
groups, labels) = loadDatasets(genDataPath)
```

```
    (time1) = measureTime(time1)

    total = np.int(yTrain.shape[0])
    totalPositive = np.int(sum(yTrain))
    print(total)
    print(totalPositive)
    (time1) = measureTime(time1)


def evaluateComposition(time1):
    print('STEP: Loading datasets')
    (xTrain,  yTrain,  xTest,  originalIndexTrain,  originalIndexTest,
groups, labels) = loadDatasets(genDataPath)
    (time1) = measureTime(time1)

    print('STEP: Loading binary classification model')
    classifiers = loadModel(modelPath)
    (time1) = measureTime(time1)

    print('STEP: Prepare data')
    (orderedXTrain,     orderedIndexTrain,     orderedYTrain)     =
orderByDevice(xTrain, originalIndexTrain, yTrain)
    (orderedXTest,    orderedIndexTest)    =    orderByDevice(xTest,
originalIndexTest)

    print('STEP: Loading pairwise model')
    pairwiseClassifiers = loadModel(pairwiseModelPath)
    (time1) = measureTime(time1)

    print('STEP: Predicting')

    resultsTrain = np.zeros(xTrain.shape[0])
    resultsTest = np.zeros(xTest.shape[0])
    index = 0
    for classifier in classifiers:
        index += 1
        print(index)

        pTrain = predictXGBoost(xTrain, classifier)
        resultsTrain = resultsTrain + pTrain
        pTest = predictXGBoost(xTest, classifier)
        resultsTest = resultsTest + pTest

    resultsTrain = resultsTrain / np.float_(len(classifiers))
    resultsTest = resultsTest / np.float_(len(classifiers))

    (resultTrain,   thresholdTrain)   =   bestSelection(resultsTrain,
originalIndexTrain, np.array([1.0, 0.96]), groups, 0)
    (resultTest,    thresholdTest)    =    bestSelection(resultsTest,
originalIndexTest, np.array([1.0, 0.96]), groups, 0)

    index = 0
    for classifier in pairwiseClassifiers:
        index += 1
        print(index)

        pTrain = predictXGBoost(orderedXTrain, classifier)
        resultsTrain = resultsTrain + pTrain
        pTest = predictXGBoost(orderedXTest, classifier)
        resultsTest = resultsTest + pTest

    resultsTrain = resultsTrain / np.float_(len(pairwiseClassifiers))
    resultsTest = resultsTest / np.float_(len(pairwiseClassifiers))

    (resultPairwiseTrain,              thresholdTrain)              =
bestSelection(resultsTrain,    orderedIndexTrain,    np.array([1.0,
0.84]), groups, 0)
    (resultPairwiseTest, thresholdTest) = bestSelection(resultsTest,
orderedIndexTest, np.array([1.0, 0.84]), groups, 0)

    for deviceId, cookieSet in resultPairwiseTrain.items():
        id = np.int(deviceId)
        otherSet = resultTrain[id]
        otherSet = (otherSet | cookieSet)
        resultTrain[id] = otherSet

    for deviceId, cookieSet in resultPairwiseTest.items():
        id = np.int(deviceId)
        otherSet = resultTest[id]
        otherSet = (otherSet | cookieSet)
        resultTest[id] = otherSet

    F05 = calculateF05(resultTrain, labels)
    print("Train F0.5", F05)

    F05 = calculateF05(resultTest, labels)
    print("Test F0.5", F05)

    (time1) = measureTime(time1)
```

4. TrainLibrary

```
import numpy as np
import os
import sklearn
import pickle
import xgboost as xgb


def bestSelection(predictions, originalIndex, values, groups, beta):

    result = dict()
    threshold = dict()
    for deviceId, cookieDict in originalIndex.items():

        resultCookieSet = set()
        maxval = 0.0
        cookiesList = list(cookieDict.keys())

        scores = np.zeros(len(cookiesList))

        for i in range(len(cookiesList)):
            scores[i] = predictions[cookieDict[cookiesList[i]]]

        ordering  =  sorted(range(len(scores)),  key=lambda  x:  -
scores[x])

        if len(cookiesList) > 0:
            if groups.get(cookiesList[ordering[0]], -100) != -100:
                maxval = scores[ordering[0]]
                resultCookieSet      =      (resultCookieSet      |
groups[cookiesList[ordering[0]]])

        for i in range(len(values)):
            if i <= len(resultCookieSet):
                if i < len(cookiesList) and (i < len(values)):
                    tam1  =  len(groups.get(cookiesList[ordering[0]],
set()))
                    tam2  =  len(groups.get(cookiesList[ordering[i]],
set()))
                    if tam1 > 1 & tam2 == 1:
                        if  scores[ordering[i]]  >  maxval*(values[i]-
0.15):
                            resultCookieSet   =   (resultCookieSet   |
groups.get(cookiesList[ordering[i]], set()))
                    elif tam1 > 1 & tam2 > 1:
                        if          scores[ordering[i]]           >
maxval*(values[i]+0.1):
                            resultCookieSet   =   (resultCookieSet   |
groups.get(cookiesList[ordering[i]], set()))
                    elif tam1 == 1 & tam2 == 1:
                        if scores[ordering[i]] > maxval*(values[i]):
                            resultCookieSet   =   (resultCookieSet   |
groups.get(cookiesList[ordering[i]], set()))

        result[deviceId] = resultCookieSet
        threshold[deviceId] = maxval
    return result, threshold


def fullTraining(xTrain, yTrain, xTest, originalIndexTrain, groups):

    nFolds = 6
    nRounds = 100
    kFold      =      sklearn.model_selection.KFold(n_splits=nFolds,
random_state=0)

    resultadosVal = np.zeros(len(yTrain))

    (tamTST, dTST) = xTest.shape
    resultadosTST = np.zeros(tamTST)

    classifiers = list()

    iteration = 0
    mapper = list(originalIndexTrain.keys())
    for (train, test) in kFold.split(mapper):

        iteration = iteration + 1
        Originaltmp = dict()
        print("Training bagger {0} of {1}".format(iteration, nFolds))

        trainIndexes = list()
        testIndexes = list()
        trainDevices = list()
        testDevices = list()

        for i in train:
            deviceId = mapper[i]
            trainDevices.append(deviceId)

trainIndexes.extend(originalIndexTrain[deviceId].values())

        for i in test:
            deviceId = mapper[i]
            testDevices.append(deviceId)

testIndexes.extend(originalIndexTrain[deviceId].values())
            Originaltmp[deviceId] = originalIndexTrain[deviceId]

        trainIndexes = np.array(trainIndexes)
        testIndexes = np.array(testIndexes)

        xValueTrain = xTrain[trainIndexes, :]
        xValueTest = xTrain[testIndexes, :]

        yValueTrain = yTrain[trainIndexes]
        yValueTest = yTrain[testIndexes]

        bst = trainXGBoost(xValueTrain, yValueTrain, nRounds, 0.10,
xValueTest, yValueTest)

        classifiers.append((bst, trainDevices, testDevices))

        predictedTest = predictXGBoost(xValueTest, bst)
        resultadosVal[testIndexes] = predictedTest

        (validat, thTR) = bestSelection(resultadosVal, Originaltmp,
np.array([1.0]), groups, 1)

        pTST = predictXGBoost(xTest, bst)

        resultadosTST = resultadosTST + pTST

    resultadosTST = resultadosTST / np.float_(nFolds)
    return resultadosVal, resultadosTST, classifiers


def trainXGBoost(xtr, ytr, rounds, eta, xtst, ytst):
    xgmat = xgb.DMatrix(xtr, label=ytr)
    xgmat2 = xgb.DMatrix(xtst, label=ytst)
    param = {'eta': eta,
             'max_depth': 10,
             'subsample': 1.0,
             'nthread': 12,
             'min_child_weight': 4,
             'gamma': 5.0,
             'colsample_bytree': 1.0,
             'silent': 1,
             'objective': 'binary:logistic',
             'eval_metric': 'error'}

    watchlist = [(xgmat, 'train'), (xgmat2, 'test')]
    num_round = rounds
    bst = xgb.train(param, xgmat, num_round, watchlist)
    return bst


def fullTrainingPairwise(xTrain, yTrain, xTest, originalIndexTrain,
groups):

    nFolds = 6
    nRounds = 200

    kFold      =      sklearn.model_selection.KFold(n_splits=nFolds,
random_state=0)

    resultadosVal = np.zeros(len(yTrain))

    (tamTST, dTST) = xTest.shape
    resultadosTST = np.zeros(tamTST)

    classifiers = list()
    iteration = 0
    mapper = list(originalIndexTrain.keys())
    for (train, test) in kFold.split(mapper):
        iteration = iteration + 1
        originalTestTemp = dict()
        print("Training bagger {0} of {1}".format(iteration, nFolds))

        trainIndexes = list()
        testIndexes = list()
        trainDevices = list()
        testDevices = list()
        groupTrain = list()
        groupTest = list()

        for i in train:
            deviceId = mapper[i]
            trainDevices.append(deviceId)

groupTrain.append(len(originalIndexTrain[deviceId].values()))

trainIndexes.extend(originalIndexTrain[deviceId].values())

        for i in test:
            deviceId = mapper[i]
            testDevices.append(deviceId)

groupTest.append(len(originalIndexTrain[deviceId].values()))

testIndexes.extend(originalIndexTrain[deviceId].values())
            originalTestTemp[deviceId]                             =
originalIndexTrain[deviceId]

        trainIndexes = np.array(trainIndexes)
        testIndexes = np.array(testIndexes)
        groupTrain = np.array(groupTrain)
        groupTest = np.array(groupTest)

        xValueTrain = xTrain[trainIndexes, :]
        xValueTest = xTrain[testIndexes, :]
        yValueTrain = yTrain[trainIndexes]
        yValueTest = yTrain[testIndexes]

        bst = trainXGBoostPairwise(xValueTrain, yValueTrain, nRounds,
0.2, xValueTest, yValueTest,
                                   groupTrain, groupTest)

        classifiers.append((bst, trainDevices, testDevices))

        predictedTest = predictXGBoost(xValueTest, bst)

        resultadosVal[testIndexes] = predictedTest

        pTST = predictXGBoost(xTest, bst)
        resultadosTST = resultadosTST + pTST

    resultadosTST = resultadosTST / np.float_(nFolds)
    return resultadosVal, resultadosTST, classifiers


def orderByDevice(xValues, originalIndex, yValues = None):
    added = 0
    triples = np.zeros([xValues.shape[0], 3])
    orderedIndex = dict()
    for deviceId, cookieDict in originalIndex.items():
        orderedIndex[deviceId] = dict()
        for cookieId, index in cookieDict.items():
            triples[added, 0] = deviceId
            triples[added, 1] = cookieId
            triples[added, 2] = index
            added += 1

    ordering   =   sorted(range(triples.shape[0]),   key=lambda   x:
triples[x, 0])
    orderedXValues = np.zeros(xValues.shape)

    if yValues is not None:
        orderedYValues = np.zeros(yValues.shape)
        for i in range(xValues.shape[0]):
            ind = np.int(triples[ordering[i], 2])
            orderedYValues[i] = yValues[ind]
            orderedXValues[i, :] = xValues[ind, :]
            orderedIndex[np.int(triples[ordering[i],
0])][np.int(triples[ordering[i], 1])] = i
        return orderedXValues, orderedIndex, orderedYValues

    else:
        for i in range(xValues.shape[0]):
            ind = np.int(triples[ordering[i], 2])
            orderedXValues[i, :] = xValues[ind, :]
            orderedIndex[np.int(triples[ordering[i],
0])][np.int(triples[ordering[i], 1])] = i

        return orderedXValues, orderedIndex


def   trainXGBoostPairwise(xtr,   ytr,   rounds,   eta,   xtst,   ytst,
groupTrain, groupTest):
    xgmat = xgb.DMatrix(xtr, label=ytr)
    xgmat2 = xgb.DMatrix(xtst, label=ytst)
    param = {'eta': eta,
             'max_depth': 10,
             'subsample': 1.0,
             'nthread': 12,
             'min_child_weight': 4,
             'gamma': 5.0,
             'colsample_bytree': 1.0,
             'silent': 1,
             'objective': 'rank:pairwise',
             'eval_metric': 'error'}

    xgmat.set_group(groupTrain)
    xgmat2.set_group(groupTest)
    watchlist = [(xgmat, 'train'), (xgmat2, 'test')]
    num_round = rounds
    bst = xgb.train(param, xgmat, num_round, watchlist)
    return bst


def predictXGBoost(x, bst):
    xgmat = xgb.DMatrix(x)
    return bst.predict(xgmat)

def calculateF05(results, target):
    BetaQ = 0.5 * 0.5

    F05 = list()

    for k in results.keys():
        correctCookies = results[k]
        predictedCookies = target[k]

        tp = np.float_(len(correctCookies & predictedCookies))
        fp = np.float_(len(correctCookies) - tp)
        fn = np.float_(len(predictedCookies) - tp)

        if tp > 0:
            p = tp / (tp + fp)
            r = tp / (tp + fn)
        else:
            p = r = 0

        if p * r > 0.0:
            f = (1.0 + BetaQ) * p * r / (BetaQ * p + r)
        else:
            f = 0.0

        F05.append(f)
    return np.mean(F05)


def loadDatasets(genDataPath):

    indexTrainFile = genDataPath + 'originalIndexTrain.pkl'
    f = open(indexTrainFile, "rb")
    originalIndexTrain = pickle.load(f)
    f.close()
    indexTestFile = genDataPath + 'originalIndexTest.pkl'
    f = open(indexTestFile, "rb")
    originalIndexTest = pickle.load(f)
    f.close()

    indexTestFile = genDataPath + 'groups.pkl'
    f = open(indexTestFile, "rb")
    groups = pickle.load(f)
    f.close()

    indexTestFile = genDataPath + 'labels.pkl'
    f = open(indexTestFile, "rb")
    labels = pickle.load(f)
    f.close()

    xTrain = np.load(genDataPath + 'xTrain.npy')
    yTrain = np.load(genDataPath + 'yTrain.npy')
    xTest = np.load(genDataPath + 'xTest.npy')

    return     xTrain,     yTrain,     xTest,     originalIndexTrain,
originalIndexTest, groups, labels


def saveModel(modelPath, classifiers):
    d = os.path.dirname(modelPath)

    if not os.path.exists(d):
        os.makedirs(d)

    modelFile = modelPath + os.path.sep + 'model.pkl'

    f = open(modelFile, "wb")

    pickle.dump(len(classifiers), f)

    nClassifier = 0

    for (classifier, indtr, indtst) in classifiers:
        classifier.save_model(modelPath     +     os.path.sep     +
str(nClassifier) + '.model')
        nClassifier = nClassifier + 1
    f.close()


def loadModel(modelpath):
    modelfile = modelpath + 'model.pkl'

    f = open(modelfile, "rb")

    nclassifier = pickle.load(f)

    f.close()
    classifiers = list()

    for i in range(nclassifier):
        classifier = xgb.Booster({'nthread': 12})
        classifier.load_model(modelpath + str(i) + '.model')
        classifiers.append(classifier)

    return classifiers
```

5. Analytics

```
import re
import matplotlib.pylab as plt

def calcConformity(ipFile, dictDevice, dictCookie, IPs, labels):

    deviceIPS = dict()
    cookieIPS = dict()

    with open(ipFile) as fp:
        fp.readline()

        ind = 0

        for line in fp:

            if ind % 200000 == 0:
                print(ind)

            matchObj  =  re.match(r'([a-zA-Z0-9_]*),([0-1]),{([(a-zA-
Z0-9(),\-_]*)}', line, flags=0)
            ips    =    re.findall(r'(\w*,\w*,\w*,\w*,\w*,\w*,\w*)',
matchObj.group(3))

            ipDict = dict()
            for ip in ips:
                ipInfo = ip.split(',')
                ipGenInfo = IPs[ipInfo[0]]
                ipDict[ipInfo[0]] = ipGenInfo[0]

            if matchObj.group(2) == '0':
                deviceId = dictDevice.get(matchObj.group(1), -1)
                if deviceId > -1:
                    deviceIPS[deviceId] = ipDict
            else:
                cookieId = dictCookie[matchObj.group(1)]
                cookieIPS[cookieId] = ipDict

            ind += 1

    commonDeviceCookieIps = dict()
    for deviceId in labels.keys():
        cookieDict = dict()
        commonDeviceCookieIps[deviceId] = cookieDict

    sum = 0
    cookieNumberDict = dict()

    print('Begin association')
    for deviceId, cookieIds in labels.items():

        curNumber = cookieNumberDict.get(len(cookieIds), 0)
        curNumber += 1
        cookieNumberDict[len(cookieIds)] = curNumber

        sum += len(cookieIds)
        for cookieId in cookieIds:
            for dIp in deviceIPS[deviceId]:
                isCellular = cookieIPS[cookieId].get(dIp, -1)
                if isCellular > -1:
                    curValue                                        =
commonDeviceCookieIps[deviceId].get(cookieId, False)
                    commonDeviceCookieIps[deviceId][cookieId]       =
curValue or isCellular == 0

    print(sum)
    print('Begin calculation')
    deviceNum = len(labels)
    print(deviceNum)

    percent = 0
    nonCellularPercent = 0
    dataSize = 0
    for deviceId, cookieDict in commonDeviceCookieIps.items():
        if len(cookieDict) > 0:
            percent += 1
            dataSize += len(cookieDict)
            for value in cookieDict.values():
                if value:
                    nonCellularPercent += 1
                    break

    percent = percent / deviceNum * 100
    nonCellularPercent = nonCellularPercent / deviceNum * 100

    lists = sorted(cookieNumberDict.items())  # sorted by key, return
a list of tuples
    x, y = zip(*lists)  # unpack a list of pairs into two tuples
    plt.plot(x, y)
    plt.ylabel('Amount of devices')
    plt.xlabel('Amount of belonging cookies')
    plt.grid(True)
    plt.show()
    return percent, nonCellularPercent, dataSize


def calcTrainSize(ipFile, dictDevice, IPs):
    deviceIPS = dict()
    cookieIPS = dict()

    with open(ipFile) as fp:
        fp.readline()

        ind = 0
        for line in fp:

            if ind % 200000 == 0:
                print(ind)

            matchObj  =  re.match(r'([a-zA-Z0-9_]*),([0-1]),{(([(a-zA-
Z0-9(),\-_]*)}', line, flags=0)
            ips    =    re.findall(r'(\w*,\w*,\w*,\w*,\w*,\w*,\w*)',
matchObj.group(3))

            ipDict = dict()
            for ip in ips:
                ipInfo = ip.split(',')
                ipGenInfo = IPs[ipInfo[0]]
                ipDict[ipInfo[0]] = ipGenInfo[0]

            if matchObj.group(2) == '0':
                deviceId = dictDevice.get(matchObj.group(1), -1)
                if deviceId > -1:
                    deviceIPS[deviceId] = ipDict
            else:
                for ip in ipDict.keys():
                    curAmount = cookieIPS.get(ip, 0)
                    curAmount += 1
                    cookieIPS[ip] = curAmount

            ind += 1

    totalAmount = 0
    totalAmountNonCellular = 0

    for deviceId, ips in deviceIPS.items():
        for ip, isCellular in ips.items():
            cookieAmount = cookieIPS.get(ip, 0)
            totalAmount += cookieAmount
            if isCellular == 0:
                totalAmountNonCellular += cookieAmount

    return totalAmount, totalAmountNonCellular
```

```
def calcCoverage(labels, candidates):
    properlyFound = 0
    for deviceId, cookies in labels.items():
        containsCookie = False
        localCandidates = candidates.get(deviceId, set())
        for cookie in cookies:
            containsCookie   =   containsCookie   or   cookie   in
localCandidates
            if containsCookie:
                properlyFound += 1
                break

    coverage = properlyFound/len(labels)*100
    print("Coverage is %s" % coverage)
    return coverage
```