Cross-Device Matching for Online Advertising with Model Stacking

Dmitry Dryomov
Moscow, Russia
dremovd@gmail.com

Julia Ivkina
Moscow, Russia
i-j@yandex.ru

ABSTRACT

In this paper we describe the third-place solution for the CIKM Cup '16, DCA, Track 1. Logically our solution could be split into consecutive parts: train set transformation; edge filtration; feature generation; predicting probability of edge existence; closure. We used Random Forest, SVM, and Gradient Boosting Decision Trees models to predict if a particular edge exists. These models are ensembled by stacking.

Keywords

CIKM Cup 2016; edge prediction; entity linking; machine learning

1. INTRODUCTION

Online advertising is, perhaps, the most successful business model for the Internet known to date and the major element of the online ecosystem. Advertising companies help their clients to market products and services to the right audiences of online users. In doing so, advertising companies collect a lot of user-generated data, e.g. browsing logs and ad clicks, perform sophisticated user profiling, and compute the similarity of ads to user profiles. User identity plays the essential role in the success of an online advertising company/platform.

As the number and variety of different devices increases, the online user activity becomes highly fragmented. People check their mobile phones on the go, do their main work on laptops, and read documents on tablets. Unless a service supports persistent user identities (e.g. Facebook Login), the same user on different devices is viewed independently. Rather than doing modeling at the user level, online advertising companies have to deal with weak user identities at the level of devices. Moreover, even the same device could be shared by many users, e.g. both kids and parents sharing a computer at home. Therefore, building accurate user identity becomes a very difficult and important problem for advertising companies. The crucial task in this process is finding the same user across multiple devices and integrating her/his digital traces together to perform more accurate profiling.

	User	Team Name	F1
1	ytay017	NTU	0.4203854111 (1)
2	ls	Leavingseason	0.4166958839 (2)
3	dremovd	Дима // MLTrainings	0.4137067007 (3)
4	bendyna	Ваня MLTrainings	0.4016845811 (4)
5	namkhanhtran		0.3611029814 (5)
6	viktor		0.3298506389 (6)
7	sswt	Сергей // MLTrainings	0.3280886256 (7)
8	agrigorev	ololo	0.3251265933 (8)
9	kdqzzxxcc		0.2699745208 (9)
10	aledovsky	Alex // ML Trainings	0.1373420857 (10)

2. DATASET

For the model development, DCA released a new dataset. This dataset contains an anonymized browse log for a set of anonymized userIDs representing the same user across multiple devices. The dataset also has obfuscated site URLs and HTML titles. The task is defined in graph-theoretical terms. Trainset consists of data about nodes (userIDs at the level of devices and the corresponding clickstream logs) and a subset of known existing edges (linked users with multiple userIDs). The participants had to predict edges on the test set (identify the same user across multiple devices). The evaluation is done by calculating the ratio of correctly predicted edges using the F1 measure based on half of the edges in the test set.

In predictive analytic contests, main constraints are human time and computational resources. Part of the decisions is made based on intuition and former participants experience.

2.1 Dataset Statistics

- The number of unique tokens in titles (dictionary size): 8,485,859
- The number of unique tokens in URLs (dictionary size): 27,398,114
- The average number of events/facts per userID: 197
- The median number of events/facts per userID: 106
- The number of unique domain names: 282,613
- The number of all events for all users combined: 66,808,490

- The number of unique userIDs: 339,405
- The number of unique websites (domain + URL path): 14,148,535
- The number of users in the train set: 240,732
- The number of users in the test set (public and private leaderboard combined): 98,255
- Known matching pairs for training: 506,136
- Known matching pairs for testing (public and private leaderboard combined): 215,307

Dataset page
https://competitions.codalab.org/competitions/11171

3. SOLUTION

We use the word "node" for "uid", "edge" for a pair of linked uids.

Our solution is separated into five consecutive parts:

1. Train set transformation
2. Unsupervised edge filtration
3. Features generation
4. Edge probability prediction
5. Closure

3.1 Train set transformation

Train part is significantly bigger than test set. In numbers, train set consists of 240732 nodes with 506136 edges and test set consists of 98673 nodes with 215307 edges.

Our solution uses features based on the order of nodes. This implicitly brings a dependence on a total number of nodes in a set. Due to this dependency we want the train set to be the same size as the test set.

The train set is organized as a set of fully connected components of size 3 to 55. So, we randomly divide these components into three parts: 2 sets of a size of the test set and "rest". Rest components are not used, whereas test-size sets would be processed separately in all algorithm parts where nodes order is taken into account.

3.2 Unsupervised edge filtration

A total number of potential edges in a test set is of order 10^{10}. It seems unfeasible and unpractical to run any sophisticated algorithm on all of the edges. Filtration is an unsupervised procedure that chooses about 10^7 edges. This subset has a high recall.

Two different filtering algorithms were used. The first approach uses only fid, an identifier of some visited URL. The second approach uses visited URLs distributions. Title information wasn't used by the solution at any stage. Both these approaches are pairwise, and we discuss them in detail in Sections 3.2.1 and 3.2.2.

3.2.1 Visited URLs distribution distance

This approach is almost a reproduction of a baseline approach with several differences. We add not only first token of a URL (domain) but all the prefixes ("paths") of a visited URL.

Example:

"A/B/C/D?E" tokens chain would be treated as: {"A", "A/B", "A/B/C", "A/B/C/D", "A/B/C/D/E"}

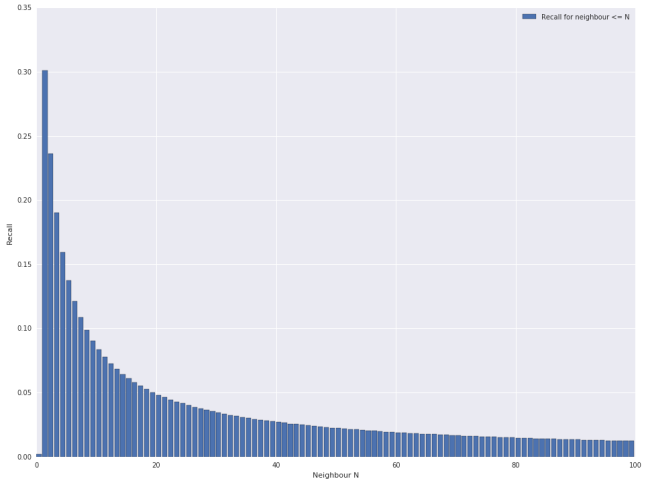
As in the baseline, TF-IDF weighting was used with per sample row-wise L_2 normalization.

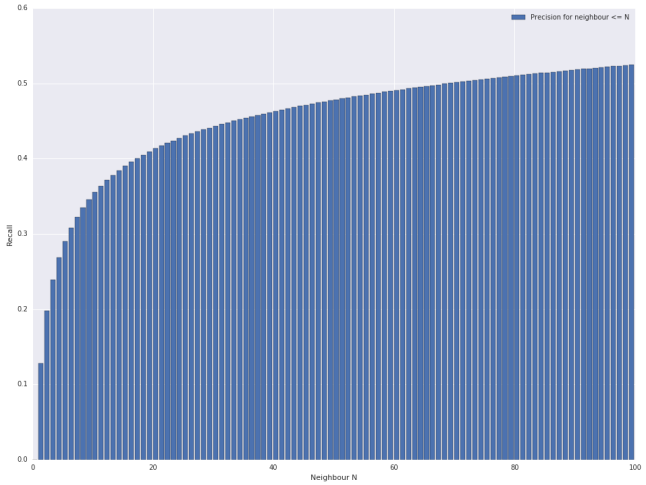
As a computational optimization, all the paths occurring less than for 2 nodes were discarded. However, they still could influence distance because of the prior normalization stage.

Brute-force nearest neighbor approach with l2 distance was used for filtering pairs.

We kept closest N neighbors for every node. Some of the edges could both presented in such set as neighbors of both ends nodes. Such "duplicates" were removed from a set. Information about neighbor number was kept as a factor for later algorithms. We kept both neighbors numbers for a "duplicated" edges as features. We used a smaller number as primary.

Cumulative precision and recall for neighbors count boundary 1..100:





Our solution keeps 15 neighbors per node.

3.2.2 Unique shared URLs counts

This approach chooses only fids that were presented in a small number of nodes.

For every pair of nodes, we calculated a count of fids that were presented in only N nodes including this pair. These counts are calculated separately per N size. By definition, N is at least 2. Counts could be efficiently computed iterating all such fids.

Example:

fid1 was presented in log history for nodes $\{A, C, D\}$;

fid2 was presented in log history for nodes $\{A, B, C, E\}$

Counts are enumerated from 1 for clarity, however for N=1 count is always 0.

For edge (A, B) counts would be $[0, 0, 0, 1, 0]$. 1 is for fid2.

For edge (A, C) counts would be $[0, 0, 1, 1, 0]$. fid1 for N=3 and fid2 for N=4.

Algorithm parameter is maximum N taken into account.

For the final algorithm maximum N is 40.

3.3 Features

Features used in solution could be separated to URL based and time-based.

3.3.1 Visited URLs features

Some filtration features were added with edges filtered:

The distance of kNN, numbers of nearest neighbor for both nodes for this edge. Count of "small-sized" fids shared in pair of nodes separately by every size. Additional feature based on the same counts is heuristic "sum": $\sum_{s=2..40} \frac{count_s}{10+s}$. Logic behind this feature is that bigger sizes should be less significant.

Additionally to the visited URLs matrix used for filtration also was used a SVD transformation of this matrix with 40 components.

Several node distances were used for nodes visited url vectors: $L_1, L_2, L_{max}, Kullback\ Leibler\ divergence, cosine, cosine-non\ normalized$.

3.3.2 Time-based features

For every node we counted how many visits were in every 2 hours intraday. Intraday distribution was L_1 normalized. Absolute values of differences for every 2 hours interval between two nodes were used as features. Also, total differences sum for all intervals was used as a feature (Equal to L_1 distance between distributions).

For every two consecutive visits, we calculated time interval. This interval has a meaning of "how much time the user spent between his/her actions". We organized buckets for these intervals. These buckets organized the way that for all nodes for every bucket we would have a similar count of events. Using these bucket distributions we would calculate features same way as for intraday distribution. Additionally, we would add statistical values (min, max, mean, median, std) for the set of all deltas of time.

For visits 'absolute' time statistical values are also calculated and added as features (i.e. mean visits time)

We calculated node time window as an interval from min(times) to max(times). These windows intersection for two nodes is an additional set of features. Both windows' sizes and difference of these sizes are used as features. "Window overlap" is calculated as the maximum time for one node time window minus the minimum time of other node time window. This values could be negative.

Count of events and difference of these counts were used as features. Count of unique fids for every node were used as features.

3.4 Edge probability prediction

3.4.1 Base models

3.4.2 Out-of-fold estimation

The train set was separated to three folds of approximately same size. All the models were built on two folds. In total, 3 sets of every model were built. All 3 sets of these model were used on the test set. Test set model estimation is the median value of correspondent models from these 3 sets.

3.4.3 Linear SVM

Linear SVM algorithm was used with sparse URL matrix. We used $sklearn.svm.LinearSVC$ which is implemented using $liblinear$. For every edge with nodes A and B vector of URL, columns were formed as a normalized dot product of URL vectors of A and B. This means that only URLs visited by both nodes at the edge are non-zero at corresponding matrix row.

Same sparse matrix was used as the only input both to Linear SVM and SVD transformation.

Due to highly different class balance for train dataset, separate SVM models were built per every nearest neighbor N. Additionally separate Linear SVM model was built for all edges not appeared in the nearest neighbor filtration. For SVM models high regularization values were used: $C \sim 10^7$

3.4.4 Random forest

scikit-learn's "Random forest" model was used without tuning with 120 trees. This model was built with all features listed in features section including SVD transformation of URL matrix.

3.4.5 Meta model

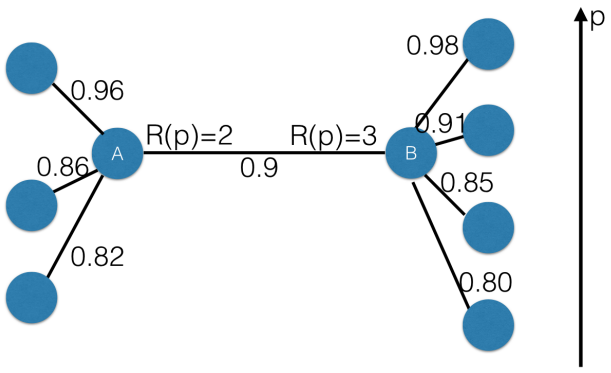
Average of 2 XGBoost models were used as final edge probability prediction. Models were built with parameters: $max\_depth : 4, eta : 0.4, objective : "binary : logistic"; num\_round = 400\ max\_depth : 4, eta : 0.3, objective : "binary : logistic"; num\_round = 500$

3.4.6 Meta features

All the features of features section were used, out-of-fold prediction of Random Forest and Linear SVM. "Rank" features calculated for all significant features and added to meta-features.

3.4.7 "Rank" features

The example below shows a process of calculation of "rank" features. We have an edge with nodes A and B. For the feature "p" we rank all the edges from the node A by this feature. Edge from A to B is ranked at position 2 among all the edges from A. Exactly the same way this edge is ranked 3 among edges from node B. So, rank features are 2 and 3. Additionally, we form features dividing rank by total edge count. These features are $\frac{2}{4}$ and $\frac{3}{5}$ in this example.



3.4.8 *Meta model combination*

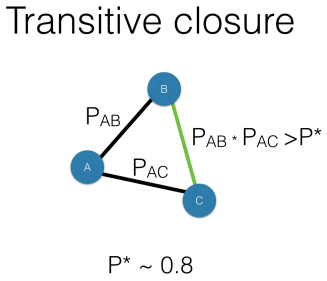
Predictions of meta models are combined in two ways: As a mean average, for the transitive closure algorithm. As a mean average edge rank number, for the rest. This is because for the rest algorithms we need only top-N edges set and no probability estimation.

3.5 Closure

3.5.1 *Transitive closure*

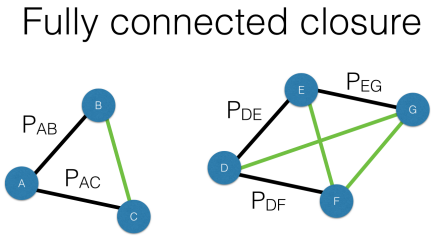
This algorithm uses probability estimation for every edge.

For every pair of edges which share one node "A" we have probabilities P_{AB} and P_{AC}. If probability $P_{AB} \cdot P_{AC} > P_{threshold}$ we mark edge BC. Used value for $P_{threshold}$ is 0.83.



3.5.2 *Fully-connected closure*

Here we using 46000 edges with the higher prediction of probability. We add all the edges to form fully connected components from edges selected. After that, we exclude all the components of size more than 60 nodes. All the edges of fully connected-components we also mark for addition.



3.5.3 *Methods combination*

Final submission is all the marked edges from both closure methods and also 97000 edges with the highest prediction of probability.

Transitive closure algorithm adding about 12000 additional edges to the 97000 edges with the highest prediction of probability.

Fully-connected closure adding about 4500 more edges on top of the both edges with the highest prediction of probability and transitive closure.

4. RESULTS

Our solution achieved the modified F1 score of 0.4137 and was ranked third in the challenge. Solution screen cast https://youtu.be/_0RdiXTceEg

Experiment results are slightly better than our best result during competition.

Table 1: Experiment results

Experiment	F1	P / 2	R
Final solution reproduction	0.4173	0.4074	0.4277
Without threshold closure	0.4158	0.4142	0.4173
Without probability closure	0.4134	0.4110	0.4158
Without closure	0.3958	0.4178	0.3760
Without "rank" features	0.4053	0.3922	0.4192
Meta model without dense	0.3975	0.3857	0.4100
Meta model without SVD	0.4120	0.4031	0.4212
Mean value of base predictors	0.4170	0.4072	0.4274
Without SVD features	0.3987	0.3874	0.4107
Without distances features	0.4040	0.3925	0.4163
Without time-based features	0.3675	0.3555	0.3803
Without intra-day features	0.4066	0.3990	0.4145
Without visits intervals features	0.4078	0.3986	0.4174
Without time-window features	0.4149	0.4058	0.4244

From experiments we can see that different set of features contributed significantly to the final solution result. The least significant set of features was based on time windows. Additional meta-features and closure procedures also were important. Also it's clear that the work is far from finished state and there is a lot of room to develop new features and closure procedures.