Classification and Learning-to-rank Approaches for Cross-Device Matching at CIKM Cup 2016

Nam Khanh Tran
L3S Research Center - Leibniz Universität Hannover
ntran@l3s.de

ABSTRACT
In this paper, we propose two methods for tackling the problem of cross-device matching for online advertising at CIKM Cup 2016. The first method considers the matching problem as a binary classification task and solve it by utilizing ensemble learning techniques. The second method defines the matching problem as a ranking task and effectively solve it with using learning-to-rank algorithms. The results show that the proposed methods obtain promising results, in which the ranking-based method outperforms the classification-based method for the task.

1. INTRODUCTION
Online advertising is to help companies market their products and services to the right audiences of online users. In doing so, advertising companies have to collect a lot of user generated data such as browsing logs and ad clicks, perform sophisticated user profiling, and compute the similarity of ads to user profiles.

However, as the number and variety of different devices increases, the online user activity becomes highly fragmented. People tend to use different devices for different purposes, for example doing work on laptops, reading documents on tablets. The same user is usually viewed independently on different 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 CIKM Cup 2016 Track 1: Cross-Device Entity Linking Challenge hosted by CodeLab from August 5th to October 5th is for tackling this problem. The competition attracted 155 registered participants, in which top-5 participants have made over 800 submissions.

Some works have been investigated to solve similar problems. [4, 1] studied on how to promote advertisement when dealing with the cross-device problem. [5, 6] focused on the problem of link prediction on a (social) graph. [8, 7]

2. DATASET
The organizers provided a collection of browsing logs and associated meta-data which are fully anonymized to allay privacy concerns and protect business sensitive information. A browsing log contains a list of events (facts) for a specific userID. Each event is mapped to a hashed URL and a hashed HTML title. In addition, a set of matching userIDs is also provided for training any supervised model. The goal of the challenge is to predict matching pairs of testing userIDs, in which the userIDs in the training and testing set do not overlap and about 0.5% “noise” userIDs are added to the testing set. Table 1 presents some statistics of the provided dataset.

Figure 1 and Figure 2 show the distribution of number of pairs given the number of facts and domains shared between the userID pairs in the training set, respectively. There is only about 1.5% of pairs does not share any facts, and about 0.4% of pairs does not access to same domains. These distributions give us a clue that even on different devices users tend to make same actions (facts) or access to the same domains.

Based on this observation, we propose a negative sampling method to create negative examples for our supervised learn-

---

Table 1: Dataset statistics

| Metric                          | Value   |
|--------------------------------|---------|
| Total users                     | 339,405 |
| Number of users in testing set  | 98,255  |
| Number of matching pairs in training set | 506,136 |
| Number of matching pairs in testing set | 215,307 |
| Number of unique tokens in titles | 8,485,859 |
| Number of unique tokens in URLs  | 27,398,114 |

---

1 The detailed description of each file can be seen on the competition website https://competitions.codalab.org/competitions/11171
3. BUILDING A TRAINING AND PREDICTING SET

3.1 Negative examples

In the training set, 506,136 matching pairs of userIDs are provided without any negative pairs, thus we need to generate a set of negative pairs for supervised learning algorithms.

In a very simple way, for each userID \( u \) in the training set, we can randomly select an userID \( v \) which are not paired with \( u \) as negative training samples, which give us a large number of negative pairs (approximately 40B). Consequently, it requires a lot of computation and not feasible for learning algorithms. Therefore, in this work we propose a simple yet effective method for creating negative samples based on the observation discussed in Section 2.

First, for each userID \( u \), we create a list of related userIDs \( L(u) \) in which each userID in \( L(u) \) shares one or more facts/domains with \( u \). Then, for each matching pair of userIDs \( (u_1, u_2) \) in the training set, we create six different negative pairs, three for the userID \( u_1 \) and three for the userID \( u_2 \). The userIDs in the negative pairs are randomly selected from \( L(u_1) \) and \( L(u_2) \), respectively; if \( L(u_i) \) is empty, they are randomly chosen from the userIDs in the training set. To this end, we obtained a total of 3.5 millions training pairs for our learning algorithms.

3.2 Prediction set

The straightforward method for generating prediction set is to pair each userID with all other userIDs in the testing set. However, the number of userIDs in the testing set are large, approximately 98 thousands, consequently we can end up with billions of possible pairs. Hence, to improve efficiency and reduce computation cost, we propose a method for generating a set of userID pairs for prediction. In practice, we observe that the method can cover almost pairs of userIDs need to predict.

As discussed in Section 2, users tend to create the same events/facts or access to the same domains across different devices such as watching videos on Youtube, accessing to social platforms Facebook/Twitter, etc. Therefore, we restrict our prediction set to only approximately 56 millions of pairs whose userIDs have one or more facts/domains in common.

4. APPROACHES

In this section, we describe two approaches for tackling the cross-device matching problem. In the first approach, we cast the problem to a binary classification task and present an ensemble learning technique to generate the prediction. In the second approach, we formulate it as a learning to rank task, and present a pairwise approach and a graph-based algorithm for generating the prediction. Before describing two approaches in details, we start by presenting our learning features.

4.1 Learning features

For each pair of userIDs, we first extract a set of features based on the provided browsing logs and associated metadata: fact-based, domain-based, title-based, time-based and hybrid features.

4.1.1 Fact-based features

The first type of feature is generated based on the fact/event information. Each user is considered as a document and the facts that she/he made are considered as words of the document. Then, jaccard, tf-idf similarities are computed as learning features.

In addition, we extract topic distributions of documents (users) by applying the topic modeling method i.e. Latent Dirichlet Allocation (LDA)\(^2\). Each user is now represented by a distribution of topics, then some similarity measures including Hellinger, cosine, jaccard are calculated as additional features.

Furthermore, we also employ doc2vec\(^3\) to represent each user as a vector of 100 dimensions and then calculate similarity measures as learning features.

4.1.2 Domain-based and title-based features

The domain-based and title-based features are extracted in a similar way as the fact-based features. Each user is considered as a document, however the words of the document are now represented by domains and titles instead of facts.

\( ^2 \)http://mallet.cs.umass.edu/topics.php
\( ^3 \)https://radimrehurek.com/gensim/models/doc2vec.html
In Layer 1, the training examples in Layer 0 are first split into learning and validation sets in which 2/3 examples are used for learning, and 1/3 for validation. The parameters of the algorithms in Layer 1 are learned using the K learning, and 1/3 for validation. The parameters of the algorithms in Layer 2. The parameters are again tuned using the examples in the validation set with the K folds strategy. The best parameter values are then used to calculate prediction scores for the pairs in the prediction set and then generate the top-N pairs for the submission.

In Layer 1, we explore several supervised learning algorithms including Neural Network (NN), Extreme Gradient Boosting (xgboost) and Random Forest, and in Layer 2, we make use of xgboost to compute the final prediction scores.

4.3 Prediction as ranking

Problem Given an userID \( u \), rank userID \( v \) based on the probability of referring to the same user. In this work, we use LambdaRank, the state-of-the-art learning-to-rank algorithm, to tackle the ranking task. To obtain the final prediction, we propose a simple yet effective graph-based algorithm using the ranking scores obtained from LambdaRank.

Method In a pairwise learning-to-rank algorithm [2], we accumulate cost only between pairs of items in the same list. The cost \( L_{ij} \) associated with the assigning scores \( s_i \) and \( s_j \) to examples \( i, j \) respectively is:

\[
L_{ij} = \log(1 + \exp(-f_{ij}(s_i - s_j)))
\]

where \( f_{ij} = 1 \) when \( y_i > y_j \), \( f_{ij} = -1 \) when \( y_i < y_j \) and \( f_{ij} = 0 \) when \( y_i = y_j \).

Minimizing the cost on the training data, and applying to validation/testing set gives us a ranking of userIDs (documents) given an userID (query) with a associated scores.

In order to use learning-to-rank algorithms, we need to create datasets with query and documents information. To obtain this purpose, we treat each userID \( u \) in the training/prediction set as query, and other userIDs which are associated with \( u \) as documents. Then, we employ LambdaRank to learn the ranking score of the documents (userIDs) for a given query (userID).

For each userID \( u_i \), the learning-to-rank algorithm returns a set of weighted userIDs \( S(u_i) = \{w_i^1, w_i^1, \ldots, (w_i^p, w_i^p)\} \) where \( w_i^j \) is a possible associated userID of \( u_i \) and \( w_i^j \) is the association weight. The task now is to generate the most probable matching pairs from this ranking. To tackle this task, we propose two different methods.

Rank 1 Sort the pairs of userIDs \((u, v)\) based on its ranking score, and take top-N pairs with the highest scores.

Rank 2 The Rank 1 method is not optimal in the sense that a userID can be associated with zero or hundreds of userIDs. Noted that every userID in the test set is connected with some other userIDs (constraint 1). In addition, if we represent each userID as a node, and two userIDs in a pair forms an edge, we obtain a userID graph. The userIDs which refer to the same user form a connected component in the graph.

Figure 4 shows the distribution of the number of connected components given the number of nodes in the training graph. Here, we also want the predictions to follow the same distribution as in the training graph (constraint 2). Therefore, we propose a simple yet effective algorithm as shown in Algorithm 1 to generate predictions satisfying the constraints.

4.4 Implementation

The scripts to generate negative examples, prediction pairs and extract learning features as described in Section 3 are...
Figure 4: Distribution of the number of connected components over nodes in the training userID graph

Algorithm 1: Greedy algorithm for generating final predictions

Input: a list of triples \( \{u, v, w\} \), number of pairs to return \( N \)

Output: top-\( N \) most probable pairs.

Sort: \( L \leftarrow \{u, v, w\} \) according to weight \( w \)

P = list() // most probable pairs

F = map() // number of associated nodes

for \( k \) in \( \{1, 2, 51\} \) do

for \( (u,v,w) \) in \( L \) do

if \( F(u) < k \) and \( F(v) < k \) then

add \((u,v)\) → P

increase \( F(u) \)

increase \( F(v) \)

end

end

end

Return top-\( N \) pairs in P

implemented using python\(^4\). For learning algorithms, we make use of several libraries including scikit-learn\(^5\), keras\(^6\), and xgboost\(^7\).

5. RESULT

Table 2 presents the results of classification methods on our validation set. In the validation set, there is 167,055 positive examples, thus the results reported in Table 2 are computed at top 167,055. We do not care about tuning the number of prediction pairs at this point. It can be seen that xgboost obtains the highest scores, while neural network and random forest achieve slightly lower results. Table 2 also shows that the stacking method outperforms all isolated classification algorithms in the Layer 1, which indicates the effectiveness of stacking learning technique for the classification task.

Table 3 shows the results of the classification methods on the provided validation set. It shows a similar pattern to the results on our validation set, in which the stacking method outperforms all individual learning algorithms. However, the performance difference is not large as it is shown on our validation set.

We develop the learning-to-ranking approach when the validation phase was closed, consequently we can tune our parameters only on our validation set. Therefore, we can only report the results during the testing phase.

Table 4 shows the results of our methods among top-5 participants, in which the learning-to-rank method outperforms the classification methods and the greedy algorithm rank\(_2\) obtains better results than the rank\(_1\) algorithm.

In addition, as we can submit any number of prediction pairs, tuning this number can boost the rank on the testing leaderboard, the rank\(_2\)-tuned algorithm.

6. CONCLUSIONS

We have presented two different approaches for tackling the cross-device matching problem. In the classification-based approach, the ensemble learning technique shows improvements over the isolated classification algorithms. The ranking-based method with a proposed greedy algorithm for generating the predictions outperforms the classification-based method. Furthermore, we also show that tuning the number of prediction pairs can increase the \( F_1 \) scores and obtain a higher rank in the final ranking.
7. REFERENCES

[1] K. J. Bannan and G. Cohane. Cross-device advertising: How to navigate mobile marketing next big opportunity. In The Ad Age Content Strategy Studio, 2014.

[2] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical report, 2010.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In KDD ’16, 2016.

[4] G. M. Lee, J. Lee, and A. B. Whinston. Matching mobile applications for cross promotion. In Proceedings of the Workshop on e-Business, 2014.

[5] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In CIKM ’03, 2003.

[6] R. N. Lichtenwalter, J. T. Lussier, and N. V. Chawla. New perspectives and methods in link prediction. In KDD ’10, 2010.

[7] R. C. Mark Landry, Sudalai Rajkumar S. Multi-layer classification: Icdm 2015 drawbridge cross-device connections competition. In ICDMW ’15, 2015.

[8] J. Walthers. Learning to rank for cross-device identification. In ICDMW ’15, 2015.