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 describe our classification and learning-to-rank approaches for tackling the problem of cross-device matching for online advertising at CIKM Cup 2016. The former method considers the matching problem as a binary classification task while the latter one transforms it to a ranking task. The results show that the learning-to-rank method provides an improvement over the binary classification for the task.

1. INTRODUCTION

The CIKM Cup 2016 Track 1: Cross-Device Entity Linking Challenge was hosted by CodeLab from August 5th to October 5th. The competition attracted 155 registered participants, in which top-5 participants have made over 800 submissions in less than 2 months.

The goal of the competition is to identify the same users across multiple devices (predict new edges in the anonymized userID graph) using the browse log and associated meta-data. The participants have to submit the most likely pairs of anonymized userIDs (edges in the matching graph), where both userIDs are associated with the same user. One userID might match to many userIDs because a user might have more than 2 devices.

The evaluation metric used for the challenge was the F_1 score. One important noted here is that participants can submit any number of pairs of anonymized userIDs, thus optimizing the number of prediction pairs might be needed.

In this paper, we describe our approaches for tackling this cross-device matching problem. In the first method, we formulate the as a binary classification task: Given a pair of two userIDs, classify it into two groups $\{1, 0\}$ which indicates that the two userIDs associate with the same user or different users, perspectively. In the second method, we formulate the problem as a learning-to-rank task: Given an userID, rank other userIDs according to their possibilities of referring to the same user.

CIKM Cup 2016 USA

DOI: 10.1145/1235

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

Table 1: Dataset statistics

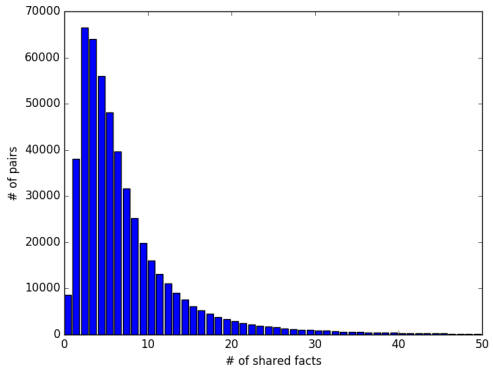

Figure 1: Distribution of shared facts over the pairs in the training set

2. DATASET

The organizers provided a collection of browse 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.[1] 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" users are added to the testing set. Table 1 presents some statistics of the provided dataset.

In Figure 1 and 2, we show the distribution of the number of shared facts and domains over the number of pairs in the training set, perspectively. There is only about 1.5% of pairs do not share any facts, and about 0.4% of pairs do not access

[1]The detailed description of each file can be seen on the competition website https://competitions.codalab.org/competitions/11171

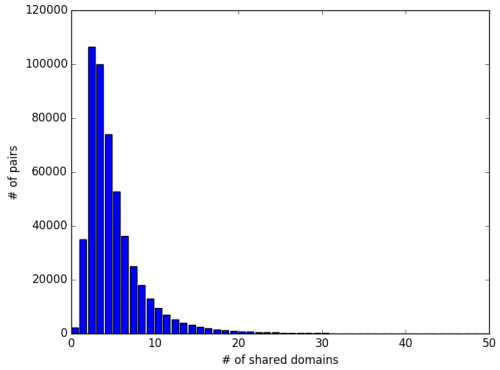

Figure 2: Distribution of shared domains over the pairs in the training set

to the same domains. The distributions give us a clue that even on different devices users tend to do the same facts or access to the same domains. Based on this assumption we create negative examples for our supervised learning method and generate a set of pairs of userIDs in the testing set for prediction.

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 user set, we can randomly select any userID which are not paired with uu in the training set as negative training samples, which results a large number of negative pairs (approximately 40B). It is not feasible for learning algorithms and computation time. Therefore, we propose a method for creating a smaller number of negative samples.

First, for each userID u, we created a list of related userIDs $L(u)$ in which each userID in $L(u)$ share one or more facts, domains with u. Then, for each matching pair of userIDs (u_1, u_2) in the training set, we created six negative samples, three for the userID u_1 and three for the userID u_2 in which which the userIDs are randomly selected from $L(u_1)$ and $L(u_2)$, respectively; if $L(u_i)$ is empty, they are randomly chosen from training userID set. To this end, we obtained a total of 3,5 millions training examples.

3.2 Prediction set

In a simple way, prediction set i.e. pairs of testing userIDs can be formed by pairing each userID with other userIDs in the testing set. However, the number of userIDs in the testing set are quite big, approximately 98 thousands, we end up with billions of possible pairs. Hence, we propose a method for generating a smaller set of pairs for prediction.

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. Thus, we restricted the prediction space only to 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 a ensembling learning technique to generate the prediction. In the second approach, we formulate it as a learning to rank problem, and present a pairwise approach and a graph-based algorithm for generating the prediction. Before describing two approaches in details, we start with presenting our learning features.

4.1 Learning features

Given a pair of userIDs, we extract a set of features based on provided browse logs and associated meta-data: *facts, domain, title, time.*

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 takes 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 distributions of topics, and 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 additional features.

4.1.2 Domain-based and title-based features

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

4.1.3 Time-based features

In the same spirit, we represent each user as a sequence of days which he/she has activities, and then calculate jaccard, rf-idf features.

In addition, we assume that users tend to use different devices at different time. For example, they use desktop/laptop in the working time such as from 9am to 12am, and mobile phone during the lunch time and their tablets in the evening. Therfore, we extract additional time-based features by calculating the distance between the time (in hour) of userIDs.

4.1.4 Hybrid features

This kind of feature takes time information and other meta-data into account, i.e. time with fact, time with domain. For example, extracting time-based features for only share facts/domains or computing fact-based features for the facts in the same day, etc.

In total, we generate 75 different learning features for our learning algorithms.

[2]http://mallet.cs.umass.edu/topics.php
[3]https://radimrehurek.com/gensim/models/doc2vec.html

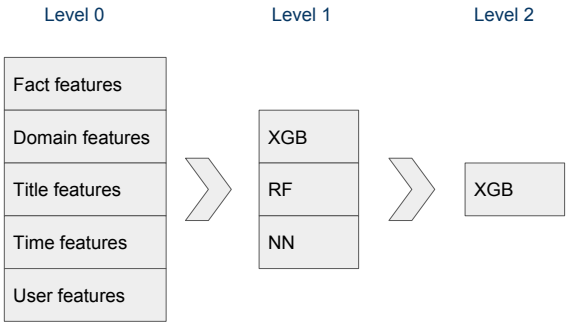

Figure 3: Ensembling learning technique for cross-device entity classification

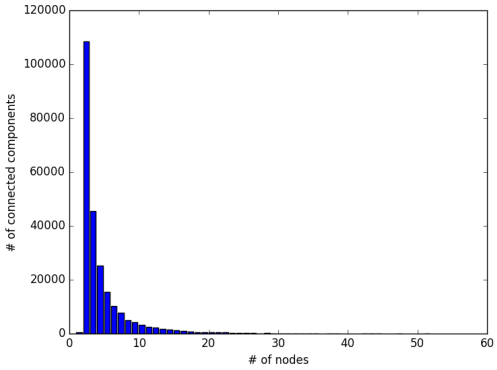

Figure 4: Distribution of connected components over nodes in the training userID graph

4.2 Prediction as classification

Problem formulation Given a pair of userIDs from prediction set, classify it into two groups $\{1, 0\}$ indicating that they are associate with the same user or different users. The final prediction is generated based on the classification scores of each pair in the prediction set.

Method Figure 3 describes an overview of our ensembling learning technique for tackling the cross-device entity matching problem. We start with 75 features extracted for each training pair of userIDs in the Layer 0. These features are used as the inputs for classification algorithms in Layer 1, whose outputs are used as the inputs for the algorithm in Layer 2.

The training samples in Layer 0 are split into learning and validating sets in which 2/3 examples are used for learning, and 1/3 for validating. The parameters of the algorithms in Layer 1 are learned using the K folds strategy on the learning set and the best configurations are tested on the examples in the validation set.

The classification scores from the algorithms in the Layer 1 are considered as the meta-features for the algorithm in Layer 2. The parameters are tuned on the examples in the validation set using the K folds method. 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 Bossting (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 formulation Given an userID, rank other userIDs based on their probabilities to refer to the same user. In order to obtain the final prediction, we propose a greedy graph-based algorithms based on the ranking scores of the learning-to-rank algorithm.

Method In a pairwise learning-to-rank algorithm [1], 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 a 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 pairs 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 scores of the documents for a given query.

For each userID u_i, we generate a set of weighted userIDs $S(u_i) = \{(v_1^i, w_1^i), ..., (v_p^i, w_p^i)\}$ where v_j^i is a possible associated userID of u_i and w_j^i is the association weight. The task now is to generate the most probable matching pairs from this ranking.

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

Rank 2 The *Rank 1* method is not optimal in the sense that a userID can be associated with zero or hundreds userIDs. Noted that (i) every userID in the test set is connected with some other userIDs. In addition, if we represent each userID as a node, and two userIDs in a pairs forms a 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 connected components over the nodes in the training graph. We want to generate the predictions which satisfies the constraint (i) and follows the same connected component distribution as in training graph. In this challenge, we propose a greedy algorithm to obtain this aim as shown in Algorithm 1.

4.4 Implementation

The scripts to generate negative samples, prediction pairs and extract learning features as described in Section 3 are implemented using python[4]. For learning algorithms, we made use of several libraries including scikit-learn[5], keras[6], and xgboost[2][7].

[4]https://github.com/namkhanhtran/cikm-cup-2016-cross-device
[5]http://scikit-learn.org
[6]https://keras.io/
[7]https://github.com/dmlc/xgboost

Algorithm 1: Greedy algorithm for generating final predictions

Input : a list of triples $\{u, v, w\}$, number of returned pairs N
Output: top-N most probable pairs.

```
Sort: L ← {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
Output top-N pairs in P
```

Method	Precision
Layer 1	
xgboost	0.927
neural network	0.911
random forest	0.904
Layer 2	
xgboost	0.932

Table 2: Performance of classification methods on our validation set

5. RESULT

Table 2 presents the results of the 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. It can be seen that xgboost obtains the highest scores, while neural network and random forest achieve slightly lower results. Table 2 also shows the effectiveness of stacking/ensembling technique which outperforms all individual algorithms in the Layer 1.

Table 3 shows the results of classification method on the provided validation set. It shows a similar behaviour to the results on our validation set, in which stacking method outperforms all individual learning algorithms.

For the second approach, we developed a learning-to-rank method when the validation phase is closed and tuning the parameters on our validation set. Thus we only report here the results using the testing test.

In addition, since we can submit any number of prediction pairs, tuning this number can boost the scores on the testing leaderboard. Table 4 shows the results of our methods comparing to top-4 best approaches. The learning-to-rank methods outperform the classification methods and the greedy algorithm $rank_2$ obtains better results than the $rank_1$ algorithm. Tuning the number of submission pairs also gives some benefits for the final ranking.

Method	Precision	Recall
Layer 1		
xgboost	0.208	0.416
neural network	0.195	0.390
random forest	0.172	0.343
Layer 2		
xgboost	0.209	0.417

Table 3: Performance of classification methods on the validation leaderboard (top 215,307 pairs)

Method	F1	Precision	Recall
Rank1	0.42038	0.39875	0.44449
Rank2	0.41669	0.39444	0.44160
Rank3	0.41370	0.40042	0.42790
Rank4	0.40168	0.36591	0.44520
Ours (clf)	0.29229	0.21921	0.43843
Ours ($rank_1$)	0.29680	0.22260	0.44520
Ours ($rank_2$)	0.32761	0.24570	0.49142
Ours ($rank_3$)	0.36110	0.33227	0.39540

Table 4: CIKM Cup 2016 testing leaderboard

6. CONCLUSIONS

We have presented two different approaches for tackling the cross-device matching problem. In the classification approach, the ensembling learning technique performs better than individual classification algorithms. In the learning-to-rank method, we propose a greedy algorithm which performs quite well for generating the predictions and outperforms the classification methods. In addition, we also show that tuning the number of prediction pairs can boost the F1 scores and get a higher rank in the leaderboard.

7. REFERENCES

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

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