Feature Engineering and Gradient Boosting in the Discovery Challenge

Andrey Zimovnov and Evgeny Sokolov

Yandex Data Factory

1 Introduction

Customers generate large amounts of unstructured data every day. Such data can bring a great value to a business, but extracting this value is a complex data mining problem. A typical example is a bank, where customers create transactional data on payments, visits, etc. These data can be used for scoring, fraud detection, recommender systems and lots of other tasks, but classical machine learning algorithms cannot incorporate transactional features directly — they should be transformed into numerical or categorical features first.

The ECML/PKDD 2016 Discovery Challenge is organized around two typical examples of such problems. We are given a two year trace of card payment events with geo location information and have two goals:

1. Predict 5 bank branches that the user visits most often.
2. Predict whether the user will apply for a credit card during a certain period.

Our solution consists of several steps. Firstly, we analyze the quality metrics and reduce both problems to a series of binary classification tasks. Secondly, we propose a number of features based on transactional data provided. Thirdly, we select the best model for each problem.

2 Data Description

The challenge consists of two supervised learning tasks. For both, we are given information on a set of users from 2014 for training with the actual target labels. Evaluation is done on another, completely different set of users. For these, we are given information for the first half of 2015. In both tasks we are asked to predict user behavior for the second half of 2015 for each test user [4].

2.1 User Info Data

1. `USER_ID` – Unique user ID.
2. `AGE_CAT` – Age category (a = $\leq$35, b = 36–65, c = >65).
3. `LOC_CAT` – Location category of the user (a = capital, b = city, c = village).
4. `INC_CAT` – Income category (a = low, b = medium, c = high, d = no income).
5. `GEN` – Gender (1 = male, 0 = female).

6. LOC_GEO_X – Geo info of user address is rounded to 100m.
7. LOC_GEO_Y – Geo info of user address is rounded to 100m.
8. TARGET_TASK_2 – The date when the user applied for credit card. Only given for training users.
9. C201* – Binary columns for each month. If True, the user has at least one credit card.
10. W201* – Binary columns for each month. If True, the user is categorized as "wealthy" in the system of the bank.

	USER_ID	AGE_CAT	LOC_CAT	INC_CAT	GEN	LOC_GEO_X	LOC_GEO_Y	TARGET_TASK_2	C201401	C201402	...
0	30277	a	c	b	0	857400	334900	2014.04.30	0	0	...
1	99045	b	b	b	0	699400	173300	NaN	0	0	...
2	19239	a	b	d	0	695900	170700	NaN	0	0	...
3	24396	a	b	a	0	585900	78300	NaN	0	0	...
4	111628	b	b	a	0	586000	78200	2015.06.30	1	1	...

Fig. 1. Sample of user info data.

2.2 Transaction Data

1. USER_ID – Unique user ID.
2. POI_ID – Unique shop ID.
3. CHANNEL – Type of activity (p = pos, n = webshop, b = branch).
4. DATE – Date of activity.
5. TIME_CAT – Time (rounded to a = 05–11h, b = 12–18h, c = 19–04h).
6. LOC_CAT – Event location category (a = capital, b = city, c = village).
7. MC_CAT – Anonymized market category groups (types are indexed from a to j).
8. CARD_CAT – Credit vs. debit card (c = credit card, d = debit card).
9. AMT_CAT – Amount of money spent in three categories (a = low, b = medium, c = high).
10. GEO_X – Geolocation information of the event.
11. GEO_Y – Geolocation information of the event.

3 Task 1: Predict the bank branches visited by the user

In this task we are asked to predict 5 out of 323 bank branches that the user visits most often. The solution is evaluated by the mean of the average of *cosine*@1 and *cosine*@5 for all users, where *cosine*@k is defined by

$$cosine@k = \frac{\sum_{i=1}^{k} v(p_i)\hat{v}(p_i)}{\sqrt{\sum_j v(j)^2}\sqrt{\sum_{i=1}^{k} \hat{v}(p_i)^2}},$$

	USER_ID	POI_ID	CHANNEL	DATE	TIME_CAT	LOC_CAT	MC_CAT	CARD_CAT	AMT_CAT	GEO_X	GEO_Y
0	91498	28052	p	2014-01-01	a	b	b	d	b	605119.0	58997.8
1	266177	1759	n	2014-01-01	a	NaN	j	d	b	NaN	NaN
2	202438	897	p	2014-01-01	a	b	b	d	c	698820.0	174757.0
3	109668	19939	p	2014-01-01	a	b	b	d	b	716050.0	271521.0
4	218581	13992	p	2014-01-01	a	a	j	d	c	653242.0	239511.0

Fig. 2. Sample of transaction data.

where $v(i)$ notes the actual number of visits of the branch i, $\hat{v}(j)$ notes the predicted number of visits of the branch j, and $p_1, \ldots, p_5$ ($\hat{v}(p_1) \geq \ldots \geq \hat{v}(p_5)$) are the indexes of the most visited branches by the user.

3.1 Engineered Features

User Features. For each user u we define a feature vector f_u, which is a concatenation of the following vectors:

1. Euclidean distances from the user geo location to all of the 323 bank POIs' geo locations.
2. C201* features for the first 6 months (binary columns for each month, if True, the user has at least one credit card).
3. W201* features for the first 6 months (binary columns for each month, if True, the user is categorized as "wealthy" in the system of the bank).
4. Gender, age, income, location categories one-hot encoded vectors.

User's Transactions Features. For each user u we also define a feature vector f_{ut} based on user's transactions for the first 6 months.

Each transaction is characterized by time category, location category, merchant category, card category and amount category. For any set of above features we can define a vector of counters [3] for all possible value combinations. For instance, let's take into consideration card category and amount category, which can take $(c, d, \emptyset)$ and $(a, b, c, \emptyset)$ values respectively. For them we define counters for any possible values combination (c, a), (c, b), $\ldots$, $(\emptyset, c)$, $(\emptyset, \emptyset)$. For each combination corresponding counter is incremented with every transaction having such values combination. We'll refer to the resulting vector of counters as $C(\text{card, amount})$ for the example above.

Each transaction also has a geo location, we propose to use it to count how many transactions took place next to the nearest bank POI, i.e. we increment bank POI counter each time it is the nearest to a transaction geo location. This way we acquire a vector L of counters of length 323.

The resulting vector f_{ut} is then defined as a concatenation of the following vectors:

1. $C(\text{time})$

2. C(location)
3. C(merchant)
4. C(card)
5. C(amount)
6. C(merchant, amount)
7. C(location, merchant)
8. C(time, location, amount)
9. C(card, amount)
10. C(merchant, card, amount)
11. L

3.2 Proposed method

For each user in the training set we calculate f_u and f_{ut} features and concatenate them to form a row vector x_u, this way obtaining a feature matrix X. For each user u we define a row vector y_u of length 323 that contains how many times a user visited one of the 323 bank POI's in the second 6 months of training. All the vectors y_u are stacked in the matrix Y.

We decided to go with holdout set for evaluation. We took 10% of users for testing. This scheme correlated with the public leaderboard and gave us an opportunity to make our experiments faster compared to K-fold cross-validation.

Our task was to predict the values in y_u vectors for the test users. For that task we trained 323 binary classifiers on X, where each classifier c_i, $i = 1, \ldots, 323$ predicts if the user will visit bank POI with index i or not. For each classifier c_i we took users, who visited bank POI with index i at least once as positive examples, and all the other users as negative examples. Every classifier is gradient boosted decision trees [1] implemented in xgboost. We tuned the number of trees using cross-validation on the training set, the optimal value was around 75 ± 25, but most of them had 75, so we took 75 trees for each bank POI to avoid overfitting.

At testing time we applied 323 classifiers c_i and obtained probabilities of positive class p_i from each of them. Bank branches were ranked by the predicted probabilities $\hat{y}_u = [p_1, p_2, \ldots, p_{323}]$, and this ranking was used as a final answer. Although the probabilities from different classifiers are not calibrated and it's not really a good thing to compare probabilities from different classifiers, it proved to work better than other approaches that we tried, including pairwise mode in xgboost and multi-class classification of the most visited bank POI.

4 Task 2: Upselling prediction

If the client has debit card only, the client may apply for a credit card (upselling). Out task is to predict for all users in the test set if she applies for a credit card in the testing period. The predictions are evaluated by Area Under Curve (AUC).

4.1 Engineered Features

User Features. For each user u we define a feature vector f_u, which is a concatenation of the following vectors:

1. Two geo location coordinates of the user.
2. Euclidean distance from the user to mean geo location (which is roughly the capital of the country).
3. C201* features for the first 6 months (binary columns for each month, if True, the user has at least one credit card).
4. W201* features for the first 6 months (binary columns for each month, if True, the user is categorized as "wealthy" in the system of the bank).
5. Gender, age, income, location categories one-hot encoded vectors.

User's Transactions Features. For each user u we also define a feature vector f_{ut} based on user's transactions for the first 6 months.

We use almost the same counter features that were described in section 3.1, for example C(card, amount). For this task we normalize counter vector for each set of features to have a sum of 1, which turned out to work better for this task. We add the actual sum of values to the end of the vector. We will refer to the resulting vector as CN(card, amount)

We take 200 most popular offline POIs and 40 online POIs in the dataset and create the counter vector for them as well. We increment the corresponding element of the vector every time this POI is seen in transactions. We normalize this vector to sum up to 1. We will refer to the resulting vector as PPN.

The resulting vector f_{ut} is then defined as a concatenation of the following vectors:

1. CN(time, location, merchant, card, amount)
2. PPN

4.2 Proposed method

For each user in the training set we calculate f_u and f_{ut} features and concatenate them to form a vector x_u, this way obtaining a feature matrix X. In the training dataset we have target dates across 2014 and 2015. In our model the client applies for credit card if the target date is later than the first 6 months, which is later than 2014.06.30. This way we fill Y vector with 1s and 0s.

We use 25% stratified holdout set for testing. First we perform simple feature selection based on L1-regularized logistic regression [2]. Then we train proprietary classifier MatrixNet on X. It's the implementation of gradient boosted oblivious decision trees [5] with sophisticated regularization based on leaf values weighting according to their variance.

The probabilities predicted by MatrixNet were used as answers for the task.

5 Conclusion

In this challenge we were given pretty raw data on transactions, which involved a lot of feature engineering to make it a machine learning problem. We tried a lot of other features not mentioned in the paper that turned out to be useless.

In the first task there was also a challenge of designing a method for essentially a ranking problem. We tried pairwise approach in xgboost, we tried multiclass classification for top 1 bank POI and it turned out to perform worse.

We obtained a good result on the public leaderboard for the first task, on which we spent much more time because that seemed like a more challenging problem at first sight. We have a pretty solid result for the second task, and we see a lot of deviation in the AUC based on the splitting we used, thus we guess there could be some changes in the private leaderboard for the second task.

The source codes for our solution can be found at GitHub repository https://github.com/ZEMUSHKA/ECML-PKDD-Challenge-2016.

References

1. Friedman, Jerome H.: Greedy function approximation: A gradient boosting machine. Ann. Statist. 29, 1189–1232 (2001)
2. Tibshirani, R.: Regression shrinkage and selection via the lasso. J. Royal. Statist. Soc B. 58, 267–288 (1996)
3. Misha Bilenko: Big Learning Made Easy – with Counts! https://blogs.technet.microsoft.com/machinelearning/2015/02/17/big-learning-made-easy-with-counts (2015)
4. ECML/PKDD 2016 Discovery Challenge Page https://dms.sztaki.hu/ecml-pkkd-2016
5. R. Kohavi and C.-H. Li.: Oblivious decision trees graphs and top down pruning. In Proceedings of the 14th International Joint Conference on Artificial Intelligence Vol. 2, 1071–1077 (1995)