# ECML/PKDD 2016 Discovery Challenge Bank Branch Visits and Customer Acquisition Prediction using Gradient Boosted Trees

Andy H.M. Chung

Data Scientist, Cathay Pacific - Asia Miles
andy_hm_chung@asiamiles.com

**Abstract.** We apply gradient boosted trees algorithm for branch visits prediction and credit card acquisition, based on customer information and card payment records for the past 2 years along with geolocation information, provided by OTP Bank Hungary. In this paper, we describe the algorithms and techniques used.

**Keywords:** Machine Learning, Customer Acquisition, Recommender System

## 1 Introduction

Our dataset contains 3 types of information:

1. Customer transaction time series
2. Customer information (Age, gender, address, income group, etc.)
3. All bank branches geolocation

The training data is a random sample of customers, with transaction records in 2014. The testing data is another random sample of customers, with transaction records in the first half of 2015. The bank branch visits record and the date at which they registered for a credit card have been removed from the testing data.

We considered two supervised learning tasks:

1. Predict the bank branches visited by the customer
2. Predict the likelihood to apply for a credit card for each customer

For task 1, we predicted top 5 out of 323 branches most likely to be visited $(p_{1i}, p_{2i}, ..., p_{5i})$ by customer $i$ in the second half of 2015. The evaluation metric is based on $cosine@5$ for all $N$ customers, defined by

$$cosine@5 = \frac{1}{N}\sum_{i=1}^{N}\frac{\sum_{k=1}^{5} v(p_{ki})\hat{v}(p_{ki})}{\sqrt{\sum_{k=1}^{5} v(p_{ki})^2}\sqrt{\sum_{k=1}^{5} \hat{v}(p_{ki})^2}} \tag{1}$$

where the actual and predicted number of visits of the top $k$-th branch by customer $i$ is denoted by $v(p_{ki})$ and $\hat{v}(p_{ki})$ respectively.

For task 2, we predicted the probability to apply for a credit card in the second half of 2015 for each customer. The evaluation metric is based on Area Under the Curve ($AUC$).

## 2 Algorithm

Among many machine learning algorithms, gradient boosted trees proved to have state-of-the-art performance. Based on the existing open source packages, we chose XGboost [1] as our algorithm implementation since it is computationally fast.

We will briefly review the algorithm of decision tree and gradient boosted trees.

### 2.1 Decision tree

For a decision tree $y = f(x)$ with $J$ number of leaves, it partitions the input space into $J$ disjoint regions $R_1, R_2, ..., R_j$ and modeled as

$$f_m(x) = \sum_{j=1}^{J} c_{jm} I(x \in R_{jm}) \tag{2}$$

where $m$ equals to 1, $I$ is the indicator function and the optimal choice of $\hat{c}_{jm}$ is

$$\hat{c}_{jm} = \frac{1}{|R_{jm}|} \sum_{y_i \in R_{jm}} y_i \tag{3}$$

where $|R_{jm}|$ is the total number of observations in the region $R_{jm}$.

A greedy approach for an optimal partition is determined at each split by enumerating all the features and possible split points and choosing the one such that the loss function $L$ reduction is maximized.

### 2.2 Gradient boosted trees

A gradient boosted trees $F_m(x)$ is updated as follows for a new tree $f_m$ built at iteration $m$ is

$$F_m(x) = \begin{cases} f_1(x) & \text{if } m = 1 \\ F_{m-1}(x) + s\gamma_m f_m(x) & \text{otherwise} \end{cases} \tag{4}$$

where $s$ is the shrinkage parameter and $\gamma_m$ is determined by

$$\gamma_m = \arg\min_{\gamma} \sum_i L(y_i, F_{m-1}(x_i) + s\gamma f_m(x_i)). \tag{5}$$

## 3 Data Munging and Feature Engineering

For task 1, we used 2014 1st half as training data and 2014 2nd half branch visits as target. For each customer there are 323 rows, each row represents whether the customer visited particular branch based on extracted Customer Features, Transaction Features and Branch Features.

For task 2, we merged two different time frames into one dataset:

1. 2014 1st half features, 2014 2nd half credit card registration as target
2. 2014 2nd half features, 2015 1st half credit card registration as target

For each time frame, the customer id is unique. We predicted the target using extracted Customer Features and Transaction Features.

### 3.1 Customer Features

All categorical data such as age group, gender, income group are one-hot-encoded. Address latitude and longitude are treated as continuous variables.

### 3.2 Transaction Features

The transaction time series is broken down monthly. For each month we calculated frequency of different events (high spending, morning transaction, etc.). For in-person events with geolocation information, we first calculated the distance using Euclidean distance between the customer address and the events, then we calculated the aggregate sum, standard deviation, median, maximum value of the distance grouped by customer and month.

### 3.3 Branch Features

Branch location latitude and longitude are treated as continuous variables. Derived features include popularity of the branch, distance between the branch and the customer. Branch index is not used for training.

## 4 Stacked Generalization

The idea of Stacked Generalization [2] is to use a group of first stage models to do prediction, followed by a second stage model using all the predictions from first stage models as input features.

For a N-folds stacking:

1. Split the training set in $N$ parts: $training_1, training_2, ..., training_N$ using stratified or random split
2. Predict *testing* and $training_1$ using original features from $N-1$ folds ($training_2$, $training_3$, ..., $training_N$) with first stage models
3. Similarly for $training_2, training_3, ...training_N$, the prediction for testing is the average of $N$ predictions of *testing*
4. Train the second stage model based on original features and first stage predictions

We only applied stacking on task 2 but not task 1. We tried two stacking method seperately:

1. 2-folds stacking split by two time frames

2. Stratified 10-folds stacking after merging two time frames

Since stacking index must be fixed in order to avoid leakage, we can only use either 2-folds or 10-folds stacking. Although 10-folds is a common choice as it provides minimal information loss, we finally decided using 2-folds for two reasons:

1. The training data is a combination data in two different time periods. If folds are split by time, stacking feature is built based on information from another time period, which is useful to avoid overfitting.
2. Both 2-folds and 10-folds stacking gives improvement on AUC, but 10-folds stacking shows discrepancy in improvement between Cross Validation and 30% of testing data.

In practice, a diversified combination of stacking features from different algorithms (KNN, RandomForest, SVM, NN, etc.) will outperform stacking using only one single algorithm. Since we only used XGboost here, we compensated this by building several stacking features using randomized parameters to achieve diversification.

**Fig. 1** shows the AUC perfomance of 5-folds Cross Validation and 30% of testing data using same parameters and seed. There is a strong correlation in AUC between Cross Validation and 30% of testing data. AUC is improved once stacking is applied and we included more randomized stacking feature until there is no improvement in AUC.

## 5 Hyperparameters Optimization

We applied grid search on a subset of parameters. In general, *max_depth*, *min_child_weight*, *gamma*, *lambda* and *alpha* are tuned for model complexity, *subsample*, *colsample_bytree* are tuned to avoid overfitting, *scale_pos_weight* is used to re-balance dataset. In order to speed things up we used a high *eta* (shrinkage) in the beginning and ranked parameter combinations based on Cross Validation. For the parameter combinations with top 5% ranking, we re-trained the model using a lower *eta* and selected the one with best Cross Validation result.

For stacking features we used random parameters without optimization because in practice stacking perform better when the underlying models are diversified.

Below are the parameters we used during training:

## 6 Performance and Conclusion

For task 1, we predicted the probability of visiting a particular branch for each customer using *logloss* as the local evaluation metric. We selected top 5 branches with highest probability score. The *cosine*@5 on 30% of the testing data is $\sim 0.65$.

The most important features are:

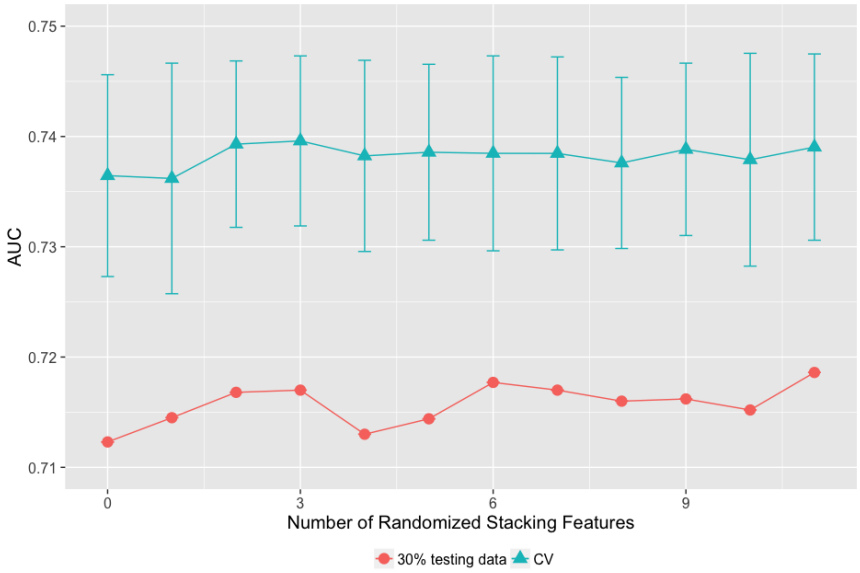


**Fig. 1.** AUC of CV result and 30% testing data using different number of stacking features. One standard deviation of the CV score is plotted via error bar.

1. Distance between the customer and the branch
2. Popularity of the branch
3. Location category of the customer

There is still a margin of improvement for task 1, since we transformed the task as a classification problem. We discarded too much information in training as we treated customer visiting a branch multiple times same as a single time.

For task 2, we predicted the probability of acquisition for each customer using *AUC* as the evaluation metric. The *AUC* on 30% of the testing data is $\sim 0.719$.

The most important features are:

1. How frequent the customer used debit card in the past 6 months
2. Whether the customer has a credit card in the last month
3. The total number of POI visited by the customer in the past 6 months
4. The income group of the customer belongs to mid range

## 7 Acknowledgments

We thank European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML/PKDD) and OTP Bank Hungary for hosting the discovery challenge.

**Table 1.** Parameters Optimization for XGboost

| Parameters | Description | Range of values | Final Choice |
|---|---|---|---|
| *eta* | Shrinkage | [0.005, 0.01, 0.05, 0.1] | 0.005 |
| *max_depth* | Maximum depth of trees | [3, 4, 5, 6, 7, 8] | 3 |
| *colsample_bytree* | Subsample ratio of columns when constructing each tree | [0.8, 0.85, 0.9, 0.95, 1] | 0.9 |
| *colsample_bylevel* | Subsample ratio of columns for each split, in each level | [0.8, 0.85, 0.9, 0.95, 1] | 1 |
| *subsample* | Subsample ratio of data when constructing each tree | [0.7, 0.8, 0.9, 1] | 1 |
| *gamma* | Minimum loss reduction required to make a further partition | [0, 0.025, 0.05, 0.1, 0.3] | 0 |
| *min_child_weight* | minimum sum of instance weight (Hessian) required to make a further partition | [1, 3, 5, 7] | 1 |
| *lambda* | L2 regularization | [0.1, 0.5, 1, 5, 10, 20] | 5 |
| *alpha* | L1 regularization | [0, 0.1, 0.5, 1, 5, 10] | 0 |

## References


1. Chen, T., Guestrin, C.: XGBoost: A Scalable Tree Boosting System (2016)
2. Wolpert, D.H.: Stacked Generalization (1992)