

Predicting Future Customers via Ensembling Gradually Expanded Trees

The LAMDAer Team*

National Laboratory for Novel Software Technology
Nanjing University, Nanjing 210093, China

Abstract. This report presents our solution to PAKDD'06 Data Mining Competition. Following a brief description on the task, we discuss the difficulties of the task and explain the motivation of our solution. Then, we propose the **GetEnsemble** (Gradually Expanded Tree Ensemble) method, which handles the difficulties via ensembling expanded trees. We evaluated the proposed method and several other methods using AUC, and found the proposed method beats others in this task. Besides, we show that how to obtain some cues on which kind of 2G customers are likely to become 3G users with the proposed method.

1 Introduction

The PAKDD'06 Data Mining Competition task is described as following:

“People are now at the time of upgrading mobile telecommunication network from the second generation (2G) to the third generation (3G). After a 3G network has been launched for a period of time, there are 2G customers switched to using 3G network. Then the company would like to make use of existing customer usage and demographic data to identify which second generation (2G) customers are likely to switch to using their 3G network. In hand, instances of 15,000 2G customers and 3000 3G customers are collected with labels. Another prediction set is provided, including instances of 6,000 customers without label. So the task is to predict the future 3G customers based on the current information, which will be evaluated in terms of how many 3G customers in the prediction set are correctly predicted.”

This report will disclose our solution to this task. The rest of the report is organized as follows. Section 2 discusses the difficulties of the task. Section 3 proposes the **GetEnsemble** (Gradually Expanded Tree Ensemble) method. Section 4 evaluates the proposed method using AUC. Section 5 shows some cues revealed by **GetEnsemble**. Finally, Section 6 concludes.

* The team comprises Yang Yu, De-Chuan Zhan, Xu-Ying Liu, Ming Li and Zhi-Hua Zhou. Correspondence: Zhi-Hua Zhou, Tel.: +86-25-8368-6268, E-mail: zhouzh@nju.edu.cn.

2 Analysis

We think the task has the following difficulties which have to be tackled:

- We believe 2G customers will switch to 3G when time goes on. This implies that the distribution of 3G customers tomorrow will be different from that today. Then, how can we deal with such time-variant target distributions?
- In the data set, the number of 2G customers are 5 times more than that of 3G customers, which makes a class-imbalance setting. Then, how can we deal with such class-imbalance?
- From the potential interest of the telephone company (e.g. its marketing plan), it may be anticipated that no potential 3G users are missed. This implies that missing a 3G user is with a bigger cost than mistakenly classifying a 2G user as a 3G user. Then, how can we deal with such different misclassification costs?
- Besides the training data set, the test data set is also given. Then, how can we exploit the test data?

At present there are many techniques which could address the above difficulties separately, such as techniques for learning with concept drift, class-imbalance learning, cost-sensitive learning, and transductive or semi-supervised learning. However, tackling these difficulties simultaneously is still an interesting challenge. The next section will present the `GetEnsemble` algorithm. It is anticipated that such an algorithm can also be useful in other scenarios.

3 The `GetEnsemble` algorithm

`GetEnsemble` is a meta-learning method. Its pseudo-code is shown in Table 1, where the sub-algorithms `DecTree`, `GeTree` and `AsymBoost` are shown in Tables 2 to 4, respectively.

As analyzed in Section 2, the concerned task is with class-imbalance and cost-sensitivity. However, it is only known that the negative class is about 5 times bigger than the positive class while the relative misclassification costs are not known. Considering that *re-weighting* is an effective scheme for decision trees¹ to deal with cost-sensitivity [3], and class-imbalance has close relationship with cost-sensitivity [8], here a weight is assigned to every training example, as shown in the Steps 3 and 4 of Table 1.

The Steps 5 to 8 of Table 1 build an ensemble of *Gradually Expanded Trees* using the labeled data set D along with the unlabeled data set D^U .

The `DecTree` algorithm in Step 8 of Table 1 reassembles the popular `J48` decision tree [6]. For the self-containness of the report, the algorithm is shown in Table 2. Readers familiar with `J48` can skip over Table 2.

¹ `GetEnsemble` uses decision tree as base learner, as shown later.

Table 1. The `GetEnsemble` algorithm

GetEnsemble

Input: D : training set; D^U : test set; T : iteration

Process:

1. $D^+ \leftarrow$ the positive subset (i.e. 3G users) of D
2. $D^- \leftarrow$ the negative subset (i.e. 2G users) of D
3. **for** $x \in D^-$ **do** $x.weight \leftarrow 1$
4. **for** $x \in D^+$ **do** $x.weight \leftarrow |D^-|/|D^+|$
5. $D^A \leftarrow D^+ \cup D^- \cup D^U$
6. **for** $t \in \{1, \dots, T\}$ **do**
7. $D' \leftarrow$ a bootstrap sample of D^A
8. $h_t \leftarrow \text{AsymBoost}(D', \text{GeTree}(\text{DecTree}), T)$

Output (soft label of x): $\frac{1}{T} \cdot \sum_{t=1}^T h_t(x)$

Table 2. The `DecTree` algorithm

DecTree

Input: D : training set

Process:

1. **if** $|D| \leq 2$, or $\forall x, y \in D : x.label = y.label$
2. **then** $p^+ \leftarrow |\{x \in D | x.label = 3G\}|/|D|$, **exit**
3. **else** $A \leftarrow$ the attribute that gives the best partition of D
4. $S = \{s_1, \dots, s_a\} \leftarrow$ partition criterions by A
5. $\{D_1, \dots, D_a\} \leftarrow$ partitions of D according to S
6. **for each** D_i **do** $h_i \leftarrow \text{DecTree}(D_i)$

Output (soft label of x):

1. **if** this is a leaf
2. **then output** p^+
3. **else if** the value of x on attribute A is not missing
4. **then** $j \leftarrow$ satisfied partition criterion s_j , **output** $h_j(x)$
5. **else output** weighted averaging of $h_i(x)$ ($i \in \{1, \dots, a\}$)

As analyzed in Section 2, the concerned task is with time-variant target distributions, which can hardly be addressed by a pure decision tree algorithm such as `DecTree`. Therefore, we propose the `GeTree` (Gradually Expanded Tree) algorithm, as shown in Table 3, which can help pure decision trees capture gradually increasing 3G users. Note that D contains labeled as well as unlabeled examples. In Step 4 the false positive rate is estimated on labeled examples of D' while in Step 5 the false negative rate is estimated on labeled examples of D , and in Step 10 some 2G or unlabeled examples will be labeled to 3G. This reflects our belief that 2G customers will switch to 3G when time goes on, as

Table 3. The GeTree algorithm

GeTree

Input: D : data set; **Learn:** decision tree learner

Process:

1. $h_f \leftarrow \text{NULL}$, $D' \leftarrow D$, $FN' \leftarrow \infty$
2. **repeat**
3. $h \leftarrow \text{Learn}(D')$
4. $FP \leftarrow |\{x \in D' | x.\text{label} = 2G \wedge h(x) = 3G\}|$
5. $FN \leftarrow |\{x \in D' | x.\text{label} = 3G \wedge h(x) = 2G\}|$
6. **if** $FP = 0$ or $FN = 0$ or $FN/FP < 1/5$
7. **then** $h_f \leftarrow h$, **exit**
8. **else if** $FN' < FN$
9. **then exit**
10. **else for** $\{x \in D' | h(x) = 3G\}$ **do** $x.\text{label} \leftarrow 3G$
11. $h_f \leftarrow h$, $FN' \leftarrow FN$
12. **end of repeat**

Output (soft label of x): $h_f(x)$

Table 4. The AsymBoost algorithm

AsymBoost

Input: D : data set; **Learn:** learner; T : iteration

Process:

1. **for** $t \in \{1, \dots, T\}$ **do**
2. normalize the weights of all instances to be summed to 1
3. $D' \leftarrow$ a bootstrap sample of D according to weights
4. $h_t \leftarrow \text{Learn}(D')$
5. $\epsilon_t \leftarrow \sum_{x \in D \wedge x.\text{label} = 3G \wedge h_t(x) = 2G} x.\text{weight}$
6. **if** $\epsilon_t > 0.5$
7. **then** $T \leftarrow t - 1$, **exit**
8. **else** $\beta_t \leftarrow \epsilon_t / (1 - \epsilon_t)$
9. **for** miss-classified 3G instance x **do** $x.\text{weight} \leftarrow x.\text{weight} \cdot \beta_t$
10. **end of for**

Output (soft label of x): $\sum_{t=1}^T \log \frac{1}{\beta_t} \cdot h_t(x)$

mentioned in Section 2. It is also worth noting that in Step 6, class-imbalance and cost-sensitivity have been considered.

The GeTree algorithm can help expand decision regions where 3G examples are densely distributed. However, it can not help on identifying 3G examples scattered in dense 2G regions. In order to address this problem, we employ the AsymBoost algorithm [4]. Roughly, here the usefulness of AsymBoost is to raise the weight of the 3G examples (in particular, 3G examples scattered in 2G

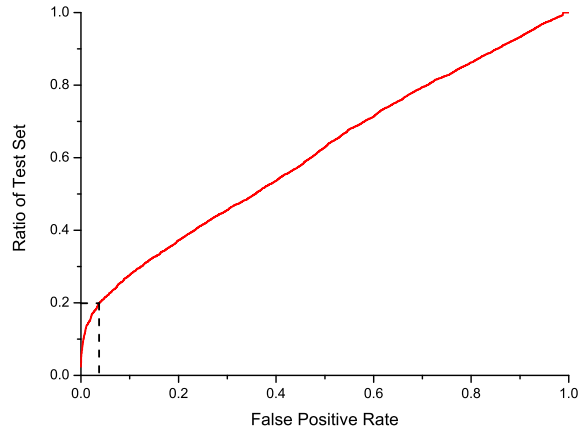


Fig. 1. To determine the output threshold for `GetEnsemble`: Ratio of test set vs. False positive rate

regions) which have been misclassified by `GeTree`, such that in the next iteration more attention will be paid to these 3G examples. For the self-containment of the report, the `AsymBoost` algorithm is shown in Table 4. Readers familiar with it can skip over Table 4.

Steps 6 to 8 of Table 1 employ `Bagging` [2] to build a ensemble of `AsymBoosted` gradually expanded trees. In other words, a `Boosting` style ensemble is used as the base learner of `Bagging`. A previous study [5] has shown that incorporating `Bagging` into `Boosting` is beneficial. Here we show that incorporating `Boosting` into `Bagging` is also beneficial. The number of iterations for `Bagging` and `AsymBoost` is set to 10, when we train the `GetEnsemble`.

Note that the output of `GetEnsemble` is not a crisp label. Instead, it is the probability that a test instance is 3G. In other words, it is a soft label. This enables the decision maker to decide the threshold of the probability beyond which an instance can be confidently put into the 3G class.

However, although soft label offers the above advantages, the Competition requires us provide crisp labels for the examples in the test set. To do so, we shall choose a probability threshold. After training a `GetEnsemble` model on all the 24,000 instances provided, including training instances with labels and test instances without labels. All the instances are sorted according to their probabilities of belonging to the 3G class in descending order. Then, we can get the *False Positive Rate* on the labeled instances if we have chosen a threshold beyond which the instances will be labeled as 3G. We define *ratio of test set* as the percentile of test instances with probability higher than a certain threshold (i.e. the percentile of test instances which will be classified to the 3G class according

Table 5. ShrinkDistribution

ShrinkDistribution

Input: D : data set; k : number of clusters; p : expanding ratio

1. $D^+ \leftarrow$ 3G instances of D
2. $D^- \leftarrow$ 2G instances of D
3. $C = C_1, \dots, C_k \leftarrow$ divide D^+ into k clusters via k -means algorithm
4. **for** $i \in \{1, \dots, k\}$ **do**
5. $c \leftarrow$ centroid of instances in C_i
6. p ratio of instances in C_i that are farthest from c are relabeled as 2G

Output: $D^+ \cup D^-$

to the threshold). Inspired by ROC curve, we plot a *Ratio of Test Set* vs. *False Positive Rate* curve², as shown in Fig. 1.

It can be observed from Fig. 1 that the curve grows sharply at the beginning with slope larger than 1, which means the distribution of these test instances are quite different from the 2G distribution in the training set. Later, the slope of the curve approaches 1, which means the distribution of those test instances is similar to the 2G distribution in the training set. The shed point is at 0.0384 of the False Positive Rate, i.e. the first 20% of the test instances should be labeled as 3G and the rest as 2G, and the corresponding probability threshold is 0.7364. Therefore, we use this threshold to derive crisp labels from the soft labels output by `GETENSEMBLE`.

4 Evaluation and Model Selection

Since we know that the target class is in expanding, typical evaluation techniques such as 10-fold cross validation could not be appropriate since they assume static distribution. In order to select a good model (we have designed other models beside `GetEnsemble`), we have designed a new evaluation scheme.

Denote the labeled data set by D and the unlabeled data set (i.e. the test set) by D^U . In our evaluation scheme, a training data set is generated from D using the algorithm shown in Table 5, which shrinks the 3G classes³. Taking $k \in \{1, 3, 5\}$ and $p \in \{10\%, 30\%, 50\%\}$, 9 labeled data sets $\{D_1^T, D_2^T, \dots, D_9^T\}$ are generated. Then, we can train a learner on $D_i^T \cup D^U$, and test it using D to get the AUC value [1].

² This curve can be plotted as follows. For every possible threshold, a *ratio of test set* can be determined on the unlabeled test set, while a *false positive rate* can be obtained on the labeled training set. Regarding them as a pair of coordinates for the threshold, a point can be drawn. Through connecting the points corresponding to different thresholds, the curve is obtained.

³ In the task we shall cope with gradually expanded target class. So, we can use a smaller target class to simulate the “current” target class, while use a bigger target class to simulate the “expanded” target class.

Table 6. AUC values of `GetEnsemble`, J48 decision tree and `AsymBoost`

Training set	<code>GetEnsemble</code>	J48	<code>AsymBoost</code>
$p = 10\%$ $k = 1$	0.9841	0.9384	0.9813
$p = 10\%$ $k = 3$	0.9812	0.9470	0.9769
$p = 10\%$ $k = 5$	0.9847	0.9347	0.9776
$p = 30\%$ $k = 1$	0.9389	0.8819	0.9245
$p = 30\%$ $k = 3$	0.9374	0.8656	0.9227
$p = 30\%$ $k = 5$	0.9421	0.8655	0.9230
$p = 50\%$ $k = 1$	0.8822	0.7821	0.8488
$p = 50\%$ $k = 3$	0.8915	0.8637	0.8586
$p = 50\%$ $k = 5$	0.8943	0.8186	0.8447

Besides `GetEnsemble`, we have also evaluated several new designs and some existing methods. Here we only present the evaluation results of `GetEnsemble`, J48 decision tree and `AsymBoost`, as shown in Table 6 where the best performance on every configuration has been boldfaced. It can be observed that `GetEnsemble` is apparently better than J48 decision tree and `AsymBoost` for all k and p values. Actually, the `GetEnsemble` method has achieved the highest AUC values in our evaluation, comparing with our other new designs. Therefore, we select it to use in this competition. That is, we use $D \cup D^U$ to train a `GetEnsemble` model and then hand in the labels it assigned to the test examples.

5 Cues

Besides strong predictive performance, we believe that a learned model can be more helpful if it could provide some comprehensible cues to the concerned task. Thus, we use the *twice-learning* scheme of the `NeC4.5` algorithm [7] to probe the trained `GetEnsemble` model.

In detail, suppose the original labeled data set is denoted by D and the unlabeled test set by D^U . We use D and D^U to train a `GetEnsemble` model at first. Then, we randomly generate 26,000 instances and put them into a D' . The instances in D and D^U are also put into D' . It is evident that D' is far bigger than D . Then, we use the trained `GetEnsemble` model to label D' , that is, the labels of all the instances in D' are now assigned by the model, which results in the data set D'' . Note that in D'' the instances are with soft labels instead of crisp labels since `GetEnsemble` makes probabilistic predictions. Considering that we are interested in customers that are not using 3G yet, we remove from D'' instances which were labeled with probability 1 by the `GetEnsemble` model. Finally, a set contains 45,106 instances with soft labels is obtained, which is then explored by a regression tree, `REPTree` [6].

Note that nominal attributes with many values will lead to a tree with many leaves, which will degrade the comprehensibility. So, before applying `REPTree`,

we has transformed nominal attributes with more than 10 values into numerical attributes.

A 3-level regression tree, shown in Fig. 2, is obtained. Each leaf is attached with a regression value, which indicates the probability this region is 3G. There are a pair of (*count/variance*) training estimation and a pair of [*count/variance*] hold out estimation, which can be treated as support and confidence.

```

44.REVPAY_PREV_CD < -100 : 0.57 (15870/0.12) [7999/0.12]
| 52.VAS_CND_FLAG = 0 : 0.34 (8248/0.08) [4207/0.07]
| | 31.BLACK_LIST_FLAG = 0 : 0.53 (4148/0.06) [2114/0.06]
| | 31.BLACK_LIST_FLAG = 1 : 0.14 (4100/0.02) [2093/0.02]
| 52.VAS_CND_FLAG = 1 : 0.82 (7622/0.06) [3792/0.06]
| | 8.CUSTOMER_CLASS = 10 : 0.85 (910/0.04) [475/0.03]
| | 8.CUSTOMER_CLASS = 3 : 0.71 (1121/0.11) [576/0.12]
| | 8.CUSTOMER_CLASS = 4 : 0.85 (889/0.04) [434/0.04]
| | 8.CUSTOMER_CLASS = 5 : 0.84 (972/0.04) [439/0.04]
| | 8.CUSTOMER_CLASS = 6 : 0.83 (923/0.04) [444/0.04]
| | 8.CUSTOMER_CLASS = 7 : 0.81 (963/0.07) [500/0.06]
| | 8.CUSTOMER_CLASS = 8 : 0.85 (960/0.04) [440/0.04]
| | 8.CUSTOMER_CLASS = 9 : 0.86 (884/0.04) [484/0.04]
44.REVPAY_PREV_CD >= -100 : 0.18 (14200/0.08) [7037/0.08]
  22.HS_AGE < 3.5 : 0.41 (2024.48/0.12) [1010.59/0.13]
  | 76.AVG_BILL_AMT < 138.57 : 0.25 (892.81/0.09) [472.72/0.1]
  | 76.AVG_BILL_AMT >= 138.57 : 0.55 (1131.67/0.11) [537.87/0.11]
  22.HS_AGE >= 3.5 : 0.14 (12175.52/0.06) [6026.41/0.06]
    7.HIGHEND_PROGRAM_FLAG = 0 : 0.12 (11535.56/0.05) [5672.49/0.05]
    7.HIGHEND_PROGRAM_FLAG = 1 : 0.51 (639.97/0.11) [353.92/0.12]

```

Fig. 2. The tree built from the data processed by **GetEnsemble**

The tree shown in Fig. 2 might give decision-makers some cues. For example, one can find that the customers in the region $REVPAY_PREV_CD < -100 \wedge VAS_CND_FLAG = 0 \wedge BLACK_LIST_FLAG = 0$ have 0.53(± 0.06) probability to be 3G customers, which are at the percent 13.88% ($\frac{4,148+2,114}{45,106}$) of all customers.

For another example, it is obviously that, the customers with $HS_AGE \geq 3.5$ and $HIGHEND_PROGRAM_FLAG = 1$ have relatively high probability, which could be explained as follows. A customer with high end program means he/she has a good income, so he/she is able to afford using the 3G network; this customer's handset is old, therefore he/she may want to have a new one. Thus, this customer may switch to the 3G network when he/she go to buy a new handset.

6 Conclusion

Based on our understanding of the PAKDD'06 Data Mining Competition task, we propose a new meta-learning algorithm **GetEnsemble** to address the difficul-

ties of the task. We have also designed a evaluation scheme for selecting potential models, which is apt to gradually expanding target classes. Besides, we have tried to obtain some comprehensible cues from the trained `GetEnsemble` model, which could be helpful to the decision-maker to understand which kind of 2G customers are likely to become 3G users.

Acknowledgements

To be added ...

References

1. Bradley, A. P.: The use of the area under the ROC curve in the evaluation of machine learning algorithms **30**(7) (1997) 1145–1159.
2. Breiman, L.: Bagging predictors. *Machine Learning* **24**(2) (1996) 123–140.
3. Ting, K. M.: An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering* **14**(3) (2002) 659–665.
4. Viola, P., Jones, M.: Fast and robust classification using asymmetric AdaBoost and a detector cascade. In: Dietterich, T. G., Becker, S., Ghahramani, Z. (eds.): *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press (2001) 1311–1318.
5. Webb, G. I.: Multiboosting: A technique for combining boosting and wagging. *Machine Learning* **40**(2) (2000) 159–196.
6. Witten, I. H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Francisco, CA: Morgan Kaufmann (2005).
7. Zhou, Z.-H., Jiang, Y.: NeC4.5: Neural ensemble based C4.5. *IEEE Transactions on Knowledge and Data Engineering* **16**(6) (2004) 770–773.
8. Zhou, Z.-H., Liu, X.-Y.: Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on Knowledge and Data Engineering* **18**(1) (2006) 63–77.