Loan Default Prediction on Large Imbalanced Data Using Random Forests

Lifeng Zhou, Hong Wang*
School of Mathematics and Statistics, Central South University
No. 22, Shaoshan South, Changsha, Hunan,410075, China
corresponding author, e-mail: wh@csu.edu.cn

Abstract

In this paper, we propose an improved random forest algorithm which allocates weights to decision trees in the forest during tree aggregation for prediction and their weights are easily calculated based on out-of-bag errors in training. Experiments results show that our proposed algorithm beats the original random forest and other popular classification algorithms such as SVM, KNN and C4.5 in terms of both balanced and overall accuracy metrics. Experiments also show that parallel random forests can greatly improve random forests' efficiency during the learning process.

Keywords: *loan default prediction, random forests, imbalanced data, parallel computing*

1. Introduction

Predicting whether a borrower would default on his/her loan is of vital importance for bankers as default prediction accuracy will have great impact on their profitability[1]. From the machine learning perspective, loan default prediction (also called credit scoring [2]) can be viewed as a binary classification problem. Previous approaches focus on prediction using ensemble methods [3] or fuzzy systems [4]. Neural networks, successfully applied to various fields [5, 6], also finds applications in the default prediction problem [7]. Most of them assume that "good" cases and "bad" cases of the data sets are equally distributed [3] or simply ignore the class distribution of data sets [4, 7].

However, most available loan defaults data sets are highly skewed, i.e. there are many cases in one class, few in another. For example, the data drawn from "Give Me Some Credit" competition on Kaggle [8] and used in this paper are also highly imbalanced. This imbalance of data causes many standard learning algorithms perform poorly on the minority class [9].

Moreover, the data sets on Kaggle.com contain hundreds of thousands of data records and this kind of large data is no rare thing in today's data-intensive age. Large amount of data makes learning or prediction of the data takes hours or even days to complete while wasting the computing power provided by the multi-core CPU system in desktop computers or the multi-CPU system in servers or clusters. To speed up the computing process, some kind of parallelism is needed as the one used in [10].

To date, there exists no specialized algorithm coping with both the imbalance and large data problem in loan default prediction. In this paper, we try to make loan default prediction on imbalanced data sets with an improved random forests approach which employs weighted majority votes in tree aggregation. The weights assigned to each tree in the forest are based on OOB(out-of-bag) errors which are easy to obtain during the forest construction process. Also, due to the fact that random forests can be paralleable in nature, we employ the foreach package in the statistical software R [11] to make random forest parallelable and greatly reduce the learning time.

The rest of the paper is organized as follows: section 2 gives a short introduction to random forests and discusses two kinds of random forests to deal with the problem in question and then proposes an improved random forests with section 3 presents the experiment results and section 4 concludes the paper.

Received July 5, 2012; Revised September 30, 2012; Accepted October 7, 2012

2. Research Method

Random forests proposed in [12] is a learning ensemble consisting of a bagging [13] of unpruned decision tree learners with a randomized selection of features at each split. The random forest algorithm outperforms many state-of-art classifiers such as SVM, ANN and does not tend to be overfitting [12]. The original random forest algorithm (for classification and regression) is as follows:

Algorithm 1: Original Random Forest

```
Input:
$T$= the given training set;
$N_{tree}$=the number of trees to be built;
$M_{try}$ = the number of predictors chosen for splitting at each node;
$S_{sampsize}$= the size of bootstrapped sample.

Process:
for ( $i_{tree}$=1; $i_{tree}$ <= $N_{tree}$ ; $i_{tree}$ ++){
        1. Select bootstrapped samples of size $S_{sampsize}$ from the given training set $T$.
        2. At each node of tree $tr_i$, select $m_{try}$ predictors randomly and determine the best split among these $m_{try}$ predictors.
        3. Construct an unpruned tree $tr_i$ using the above bootstrapped samples.
    }

Output:
        Make a decision with a majority vote from all individual trees
```

To do imbalanced prediction using random forests, [14] proposed two ways: one is cost sensitive learning which incorporates class weights into the random forests classifier, and the other is by using over-sampling methods with the minority class and/or down sampling with the majority one to balance the original data. The former is called weighted random forests while the latter balanced random forests.

One may set class weights or a cutoff threshold to construct a weighted random forest. Take the cutoff parameter for instance, the random forests classifier classifies a new instance by assigning a class probability to this instance based on the fraction of votes for that corresponding class. If the class probability is greater than the cutoff threshold, the instance is labelled positive, and negative otherwise. For imbalanced binary classification, cutoff can be set below 0.5, and doubtful instances might be labelled as the minority and the overall or average accuracy may be improved.

To create a balanced random forest, we can make use of the "*sampsize*" parameter in random forest to do stratified sampling and i.e. using a balanced training data to grow each decision tree and consequently create a balanced random forest based on these balanced trees. Alternatively, we can use wrapper sampling algorithms such as Smote [15] to create a balanced random forest. The basic idea of Smote is to randomly generate new instances of the minority class using K nearest neighbours (KNN) method and under-sampling the majority class in the training set. A new balanced training set is created by combining newly generated minority instances and the under-sampled majority ones. And this new data set is used for training the random forest classifier.

In the original random forest algorithm (Algorithm 1), each tree in the forest casts an equal vote in the prediction process and the instance in question is classified to the class label which gets the majority votes. However, due to difference in bootstrapped samples, the resulting decision tree does not have the same predictive power and this fact cannot be ignored in the prediction process. Trees with lower prediction errors should be placed higher weights.

We also note that, in Algorithm 1, the for loop depends only on N_{tree}, and each execution within the loop can be done independently. This particular parallelable feature of random forests makes it competitive in classifying large volume of data sets. We can partition the large random forest into smaller sub-random forests according to the number of CPU cores or physical processors a machine has. Then we merge the results of each random forest to make the final prediction. There are many parallel computing algorithms available, foreach package in R is an example.

Based on the above considerations, we propose the following improved algorithm:

Algorithm 2: Improved Random Forest

```
Input:
$T$= the given training set=$\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$;
$N_{tree}$=the number of trees to be built;
$M_{try}$ = the number of predictors chosen for splitting at each node;
$S_{sampsize}$= the size of bootstrapped samples;
$P$=parallel parameter depending CPU cores or physical processors

Process:
$N_t = N_{tree}$ /P
Employ foreach to generate P parallel sub-random forests{
    for ( $i_{tree}$=0; $i_{tree}$ <= $N_t$ ; $i_{tree}$ ++){
        1. Select bootstrapped samples of size $S_{sampsize}$ from $T$.
        2. At each node of tree $tr_i$, select $m_{try}$ predictors randomly and determine the best split
           among these $m_{try}$ predictors.
        3. Construct an unpruned tree $tr_i$ using the above bootstrapped samples.
        4. Record the prediction error rate $err_{pi}$ on OOB data
        }
    }
Calculate the weight $w_{pi}$ for each tree based on OOB prediction errors ($err_{pi}$) and produce a random forest $F$.

Output:
    A sample $(x,y)$ is predicated with class label $y*$ as the one receiving the weighted majority of
the classes:
    $y* = \operatorname{argmax}_y \sum_{p=1}^{P}\sum_{i=1}^{N_t} w_{pi}(tr_i(x)=y)$
```

Supprose err_{mean}, err_{min} are the mean and minimum values among all err_{pi}s respectively. The weight w_{pi} in the proposed alogirthm is calculated according to the following formula:

$$w_{pi} = \begin{cases} 1 & err_{pi} = err_{\min} \\ (err_{mean} - err_{pi})/(err_{mean} - err_{\min}) & err_{\min} < err_{pi} < err_{mean} \\ 0 & err_{pi} >= err_{mean} \end{cases}$$

When analyzing imbalanced datasets, the question of which type of random forest (a weighted random forest or a balanced one) is better in terms of prediction accuracy is still under exploration [16-18]. However, as misclassification costs are not always available, we prefer to take the balanced approach in dealing with imbalanced data sets.

3. Results and Analysis

The above Algorithm 2 has been implemented in R based on the original randomForest package [19]. We make experiments with a quad-core computer, and thus we split up the random forest into 4 sub-random forests, and then run them on different CPU cores separately. Finally, we combine these 4 sub-random forests into one big random forest for further prediction.

3.1. The dataset and evaluation criterion

The “Give me some credit” dataset on Kaggle includes a labeled dataset of 150,000 anonymous borrowers with two classes “0” and “1”. In our experiments, instances with missing values are omitted. We randomly partition the labeled data set into two sets: 80% of them are used for training and 20% for testing. Both sets are highly skewed with a class ratio of about 13:1. There are 6670 negative instances in the training set and 1687 negative instances in the test set.

We use the following abbreviations for the ease of explanation: P (\# positive instances), N (\# negative instances), TP (\# true positives), TN (\# true negatives), FP (\# false positives), FN (\# false negatives). The evaluation metrics considered in this paper are overall accuracy and balanced accuracy (average accuracy) which are calculated by the following formulas:

$$Overall\ Accuracy = \frac{TP+TN}{P+N}\ ,\ Balanced\ Accuracy = 0.5\frac{TP}{TP+FN} + 0.5\frac{TN}{TN+FP}\ .$$

3.2. Results and Discussions

We adopt the default settings of the original random forest algorithm in our proposed one where "N_{tree}=500" and additionally we set "*sampsize"* parameter to "5000". Figure 1 shows the prediction errors of all 500 trees on OOB data in the dataset. From Figure 1, we can see that different trees actually have different prediction rates and in our experiments the maximum predictor error is 11.38% and the minimum prediction error is 8.92%. The corresponding weights allocated for these trees during the test process are shown in Figure 2.

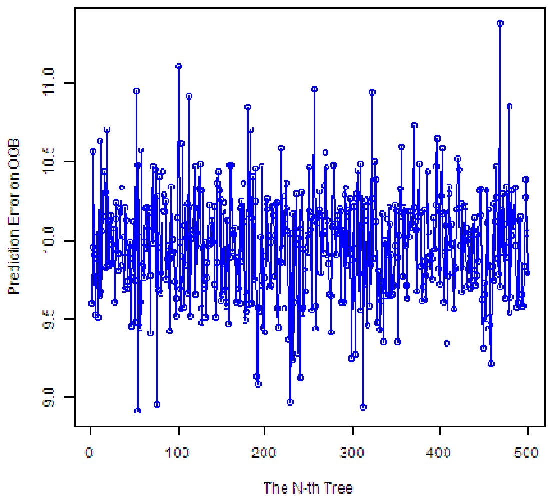

Figure 1. Prediction error rates(%) of individual decision tree on OOB data

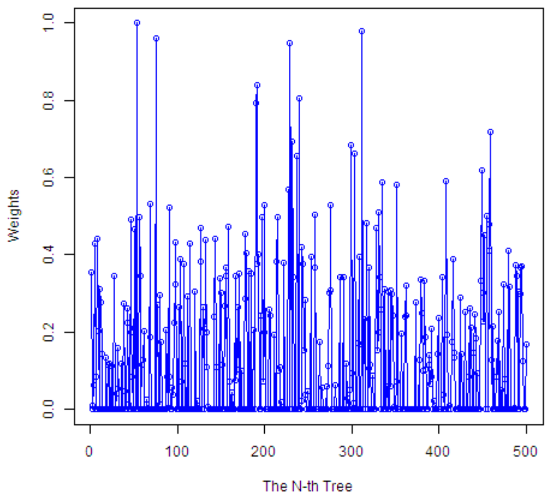

Figure 2. Weights of individual decision tree

We compare our proposed algorithm with popular classifiers such as SVM, KNN and C4.5 using related R packages. The performance of these algorithms are presented in Table 1. From the table, we may find that the proposed algorithm outweighs the standard SVM, KNN, C4.5 and original random forest in terms of both overall accuracy and balanced accuracy.

Table 1. Comparision with Other Algorithms.

	Overall Accuracy	Balanced Accuracy
SVM	0.933687	0.5403129
KNN	0.929708	0.5088449
C4.5	0.9324851	0.5772189
Original Algorithm 1	0.935842	0.5768308
Proposed Algorithm 2	0.936602	0.580837

Now, we use Algorithm 1 and Algorithm 2 to create two types of balanced random forests, one is by tuning the *sampsize* parameter and the other is by using Smote algorithm.

First, we use ten groups of *sampsize* vectors for strata sampling and the performance of two algorithms in terms of overall accuracy and balanced accuracy are shown in Table 1 and Table 2, respectively.

From Table 2 and Table 3, we find that group 7(7000, 6000) (almost balanced, down-sampling the majority class and down-sampling the minority class a little) yield a good combination of overall accuracy and balanced accuracy.

Table 2. Using *Sampsize* -Overall Accuracy

Positive	Negative	Proposed	Original
1000	6670	0.439904	0.438329
1000	4000	0.535643	0.531209
2000	2000	0.810138	0.809765
2000	5000	0.668684	0.664581
6000	1000	0.92975	0.929211
6670	6670	0.83774	0.838694
7000	**6000**	**0.856267**	**0.855728**
8000	1000	0.93348	0.933563
9000	1000	0.934972	0.934723
13000	1000	0.936049	0.935635

Table 3. Using *Sampsize* -Balanced Accuracy

Positive	Negative	Proposed	Original
1000	6670	0.677793	0.677221
1000	4000	0.718023	0.71701
2000	2000	0.780068	0.78069
2000	5000	0.759941	0.759106
6000	1000	0.669766	0.669202
6670	6670	0.77106	0.77212
7000	**6000**	**0.770055**	**0.77004**
8000	1000	0.638331	0.63673
9000	1000	0.625153	0.62502
13000	1000	0.586536	0.584943

Then, we use Smote to create a balanced random forest where parameters perc.over and perc.under control the sampling process. For instance, if perc.over =500, then for each minority instance in the original data set, 5 new examples of that class will be created. If perc.under=80, in the new dataset, majority instances with the size equal to 80% of the number of minority instances will be chosen from the original dataset.

For different groups of the two parameters, the overall accuracy and balanced accuracy are shown in Table 4 and Table 5.

Table 4. Smote Sampling-Overal Accuracy

perc.over	perc.under	Proposed	Original
100	90	0.685511	0.686795
200	90	0.770474	0.769189
100	100	0.705695	0.704576
400	100	0.84031	0.838735
600	**100**	**0.864929**	**0.864929**
800	100	0.869529	0.869695
1000	100	0.878937	0.877445
1200	100	0.881383	0.882958
200	200	0.883621	0.883413
1000	300	0.931988	0.931863

Table 5. Smote Sampling-Balanced Accuracy

perc.over	perc.under	Proposed	Original
100	90	0.763779	0.764196
200	90	0.780674	0.780531
100	100	0.7686	0.769094
400	100	0.771893	0.771869
600	**100**	**0.765118**	**0.764844**
800	100	0.756079	0.756168
1000	100	0.753188	0.751837
1200	100	0.747101	0.747674
200	200	0.757076	0.754223
1000	300	0.644107	0.643218

Table 6. Running time with and without using parallelism.

	Without Parallelism (seconds)	With Parallelism (seconds)
General	117.01	87.20
sampsize-based	166.38	82.26
Smote-based	101.05	60.17

From Tables 4 and 5, we may see that Smote-based balanced random forest generally get similar performance with *sampsize*-based one. We may notice that in the above tables 3 and 4, groups 5, which oversamples the minority class in the new set, gains the best results.

From the above Tables 2-5, we know that in most cases, the proposed algorithm gets better results than the original random forests.

Last, we use foreach package in R to implement parallelism in the above mentioned random forests and in the experiments, we set *sampsize*=5000, *mtry*=3 and *ntree*=500 and *cutoff*=c(0.5,0.5) for a general random forest (i.e. using the default settings and not tuned for imbalanced data). In two balanced random forests, we set *sampsize*=c(7000,6000), *mtry*=3 and *ntree*=500 for *sampsize*-based one; for Smote-based one, we set *sampsize*=5000, *mtry*=3 and *ntree*=500 , and perc.over=600, perc.under=100 for Smote.

Different running time of these random forests is shown in Table 6. From Table 6, we can observe that parallelism in random forests can reduce the overheads in all three cases and significantly improve the algorithms' efficiency.

4. Conclusion

In this paper, we improve the original random forest algorithm by allocating weights to decision trees and prediction decision is made based a weighted majority in the ensembling of all the trees in the forest. In our algorithm, tree weights are easily calculated based on their past performance, namely, OOB (out-of-bag) errors in training. To deal with the loan default prediction problem in question, we adopt a balanced random forests approach. We also try to speed up random forests' learning process using parallelism to cope with the large data problem.

We compare the performance of the proposed algorithm with SVM, KNN, C4.5. We also compare the proposed algorithm with the original random forest by tuning *sampsize* parameter and by using Smote sampling method. In our experiments, we use foreach package in R to realize parallelism in random forests.

Experiments show that the weighted majority approach in tree aggregation improves random forest performance in terms of overall accuracy and balanced accuracy and also outweighs many classifiers such as SVM, KNN and C4.5. In our tests, *sampsize*-tuning balanced random forest and Smote-based balanced approach get similar results in dealing with the imbalanced problem. Experiments also show that random forests with parallelism can greatly reduce the learning time random forests and this technique should be considered as a standard practice on learning large datasets.

Acknowledgment

This work was supported in part by the Fundamental Research Funds for the Central Universities of Central South University, China.

References

[1] Jozef Zurada RMB. Using Memory-Based Reasoning For Predicting Default Rates On Consumer Loans. *Review of Business Information Systems-First Quarter*. 2007; 11(1): 1-16.
[2] Hand DJ, Henley WE. Statistical Classification Methods in Consumer Credit Scoring: a Review. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*. 1997; 160(3): 523-41.
[3] Zurada J. *Data Mining Techniques in Predicting Default Rates on Customer Loans*. Proceedings of the Baltic Conference, Baltic. 2002: 1: 177-188.
[4] Odeh O, Koduru P, Featherstone A, Das S, Welch SM. A multi-objective approach for the prediction of loan defaults. *Expert Syst Appl.* 2011; 38(7): 8850-8857.
[5] Gupta M, Kumar R, Gupta R. Neural network based indexing and recognition of power quality disturbances. *TELKOMNIKA Indonesian Journal of Electrical Engineering.* 2011; 9(2): 227-326.
[6] Patricia Melin VH, Danniela Romero, Fevrier Valdez, Oscar Castillo. Genetic Optimization of Neural Networks for Person Recognition based on the Iris. *TELKOMNIKA Indonesian Journal of Electrical Engineering.* 2012; 10(2):309-20.
[7] Reddy MVJ, Kavitha B. *Neural Networks for Prediction of Loan Default Using Attribute Relevance Analysis*. Proceedings of the 2010 International Conference on Signal Acquisition and Processing. 1749237: IEEE Computer Society; 2010: 274-277.
[8] Give Me Some Credit. In: http://www.kaggle.com, 2011.
[9] He H, Garcia EA. Learning from Imbalanced Data. *IEEE Trans on Knowl and Data Eng.* 2009; 21(9): 1263-1284.
[10] Gede Indrawan BS, Saiful Akbar. Review of Sequential Access Method for Fingerprint Identification. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 2012; 10(2): 335-342.
[11] Development Core T. R: A language and environment for statistical computing. Vienna, Austria. 2012.
[12] Breiman L. Random Forests. *Mach Learn.* 2001; 45(1): 5-32.
[13] Breiman L. Bagging predictors. *Mach Learn.* 1996; 24(2): 123-140.
[14] Chen C, Liaw, Andy, Breiman, Leo. *Using Random Forest to Learn Imbalanced Data.* 2004 [cited 2012 April, 23]; Available from: www.stat.berkeley.edu/tech-reports/666.pdf.
[15] Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. *J Artif Int Res.* 2002; 16(1): 321-357.

[16] Kate McCarthy BZ, Gary Weiss. *Does cost-sensitive learning beat sampling for classifying rare classes?* Proceedings of the 1st international workshop on Utility-based data mining (UBDM '05); 2005: ACM, New York, NY, USA.
[17] Chris Seiffert TMK, Jason Van Hulse, Amri Napolitano. *A Comparative Study of Data Sampling and Cost Sensitive Learning.* Proceedings of the 2008 IEEE International Conference on Data Mining Workshops (ICDMW '08); 2008: IEEE Computer Society, Washington, DC, USA.
[18] Victoria López AF, Jose G. Moreno-Torres, Francisco Herrera. Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. Open problems on intrinsic data characteristics. *Expert Syst Appl.* 2012;39(7): 6585–608.
[19] Liaw A, Wiener M. Classification and Regression by randomForest. *R News.* 2002; 2(3): 18-22.