Multi-Class ROC Tree and Random Forest for Imbalanced Data Classification

Jiaju Yan[1], Bowen Song[2], Wei Zhu[1]

[1]Stony Brook University, Stony Brook, NY, 11794
[2]Ocean University of China, Songlinglu Road, Qingdao, Shandong, PR China, 266100

Abstract
The imbalanced class problem in classification is highly relevant in many practical scenarios such as the detection of a rare condition. One solution to this problem is to design specific algorithms incorporating the imbalanced classes in the training process of a classifier. In this paper, we propose a multi-class classification tree based on the area under the ROC curve (AUC) to resolve the imbalanced classification problem. This tree classifier aims to maximize the sum of AUC for all one versus all classifiers at the node attribute selection stage, and maximize the harmonic mean of sensitivity and specificity of all one versus all classifiers at the node threshold selection stage. The random forest framework is further applied on the ROC tree with suitable modifications. Volume Under Surface (VUS), the extension of AUC for multiple classes, is discussed in this paper and used to measure the performance of classifiers. The simulation results show that this ROC tree/forest method is superior to CART, random forest and SVM on imbalanced classification problems, while the ROC random forest performs equally well as the usual random forest and SVM on balanced classification problems.

Key Words: Imbalanced Data Classification, Multi-Class, ROC random forest, ROC Tree, ROC Surface, Volume Under ROC Surface

1. Introduction

The imbalanced data problem in classification refers to a situation where the size distribution of different classes are skewed rather than even. The usual classification algorithms do not work well on severely imbalanced data sets, and they tend to assign all observations to the majority class, rendering the minority class poorly classified.

Much works has been done to deal with the imbalanced data classification problem, with the associated approaches in 4 main categories. The first category is the pre-processing methods. Approaches in this category focuses on balancing the data before we apply classification algorithms. This includes the sampling methods as well as generating synthetic samples. A notable member in this category is the SMOTE algorithm that generates synthetic samples for the minority class [1]. The second category is the post-processing methods. Approaches in this category focuses more on modifying methods after the classifiers are built, which are mostly ensemble methods. For example, the cascade ensemble architecture in the Viola Jones algorithm is used to control the number of observations assigned to the positive class [2]. The third category is the cost sensitive learning algorithms, which assigns different weight to the false positives and the false negatives to minimize a cost function based on which [3]. The fourth category is the algorithm specific approach, which refers to classification algorithms designed to deal with imbalanced data classification problems. The ROC tree proposed in this paper belongs to this category.

The idea of using the Area Under ROC Curve (AUC) in the tree splitting method is proposed by Ferri et al. (2002) [4]. This method selects a feature and split point based on the AUC corresponding to a classifier for every potential class labelling for the induced child nodes. This is not really a ROC curve generated by varying the threshold but by exhausting every possible labelling result to form a convex hull on ROC space and choosing the edge as the ROC curve. For binary classification problem and two child leaves tree structure, this method would be inaccurate. Hossain et al. (2008) [5] conducted a study that also used the AUC measure to select a node based on the classification performance and then uses the misclassification rate to choose a split point. The misclassification rate here is the overall inaccuracy and therefore is not suitable for our imbalanced data. However, we adapt their idea of the first part, using AUC to select splitting attribute. As for the splitting threshold, we use the harmonic mean of specificity and sensitivity, which is also known as the F1 score [6]. The F1 score is more suitable to the imbalanced data set since it places equal emphasis on both classes. Therefore the combination of AUC and F1 score is the node splitting method for our binary ROC tree, which will be introduced in more details in Section 2.

In the situation of more than two classes, we adapted the One versus Rest strategy in our node splitting method [7] and expanded the F1 score to multiple classes to the multiclass ROC tree. The random forest framework with modification is applied to the multiclass ROC tree to obtain the multiclass ROC random forest, which will be introduced in Section 3.

In Section 4, we will introduce the expansion of AUC to multiple classes, the Volume Under Surface (VUS) proposed by Landgrebe et al. [8], and compare the performance of ROC random forest with SMOTE random forest on binary classification cases based on UCI repository data. Furthermore, we will also compare the performance of ROC random forest with other classification algorithms on simulated multi-class classification data.

2. Binary ROC Tree

2.1 Binary ROC Tree Structure

The structure of the ROC tree presented in this paper is similar to CART [9]. It is a binary tree where each internal nodes has 2 children. Each internal node consists of 4 elements, the splitting attribute, the splitting threshold, the left child and the right child. Each leaf contains 3 elements, the label for this leaf, the training score for this leaf and the out-of-bag score for this leaf. The tree building process is a recursive greedy strategy, in that we use the data to find the best attribute and threshold in this node. The observations that satisfy the splitting criteria are used to build the left child, while the rest utilized to build the right child. In our node building process, we randomly divide the data into training and testing sets, and use the training data to build the node and use the testing data to obtain the out-of-bag score. The leaf class is the majority class in this leaf, and the leaf score is defined to be the percentage of each class in this leaf.

2.2 Node Selection Method

The node selection method in our ROC tree can be divided to 2 parts, the node attribute selection and the node threshold selection. In the node attribute selection, we select the attribute that yields the largest AUC, while in the node threshold selection, we select the threshold that gives the largest harmonic mean of sensitivity and specificity, that is, the largest F1 score.

2.2.1 Node Attribute Selection

In the node attribute selection part, we calculate the AUC of all potential attributes and select the attribute that provides the largest AUC on current training data to be the node attribute.

2.2.2 Node Threshold Selection

In the node threshold selection, we calculate the F1 score based on the sensitivity and specificity of splits on each possible threshold. The threshold with the largest F1 score will be chosen as the node threshold.

2.3 Stopping Criteria

Note that the tree building process is a recursive process, and therefore we define some stopping criteria for the ROC tree. The stopping criteria are 1, when the node is pure or almost pure, 2, when the number of observations in this node is lower than a threshold, 3, when the maximum tree depth is reached, and 4, when the out-of-bag score differs a lot to the training score. When the stopping criterion is reached, the node will be transferred to a leaf and kept only its label and scores.

3. Multiclass ROC Tree/Random Forest

3.1 Multiclass Problem and Solution

The problem about the ROC tree in Section 2 is that it only works on binary classification problems. We solve this problem by introducing the one vs rest AUC in the node attribute selection stage and expanding the F1 score to multiple classes in the node threshold selection stage.

3.1.1 One vs Rest AUC

Suppose there are k classes, we need to compute the AUC for each class against the rest classes, which will give us k AUCs for each potential attribute. Then we calculate the sum of all one versus rest AUC for each attribute, and select the attribute with the largest sum of AUC to be the node attribute.

3.1.2 F1 Score for Multiple Classes

Here we use a simple expansion of the F1 score. In our multiple class F1 score, we calculate the sensitivity and specificity for each one versus rest AUC at each possible threshold, and then set the expanded F1 score to be the harmonic mean of all those sensitivity and specificity.

The leaf score for multiclass ROC tree is the percentage of each class in this leaf, and the leaf class is the majority class in this leaf.

3.2 Multiclass ROC Tree Algorithm

The algorithm for node attribute selection is presented below.

Algorithm 1 Node_Attribute_Selection

Input(s): $\boldsymbol{X}$, the matrix of training data; $\boldsymbol{\omega}$, the corresponding label vector; $\boldsymbol{\mathcal{A}}$, the set of column indices in $\boldsymbol{X}$ as available attributes

Output(s): $\mathcal{A}$, column indice of attribute with the highest total AUC; *max_auc*, the AUC sum of this feature for all One vs All classifiers; *AUC_sign*, the vector of signals to tell whether AUC for this class is smaller than 0.5

```
1:  max_auc=0
2:  uniq_class=unique($\boldsymbol{\omega}$)
3:  $\mathcal{A}$=0
4:  AUC_sign=rep(0, length(uniq_class))
5:  for $i$ in $\boldsymbol{\mathcal{A}}$
6:      tmp_auc=0
7:      for $j$ in 1:length(uniq_class)
8:          tmp_label=rep(0,length($\boldsymbol{\omega}$))
9:          tmp_label[which($\boldsymbol{\omega}$ == uniq_class[$j$])]=1
10:         auc_result= AUC_calculation($\boldsymbol{X}$[,i], tmp_label)
11:         if auc_result<0.5 then
12:             auc_result=1- auc_result
13:         tmp_auc=tmp_auc+ auc_result
14:     end for
15:     if tmp_auc>max_auc then
16:         max_auc=tmp_auc
17:         $\mathcal{A} = i$
18:     end if
19: end for
20: for j in 1:length(uniq_class)
21:     tmp_label=rep(0,length($\boldsymbol{\omega}$))
22:     tmp_label[which($\boldsymbol{\omega}$ == uniq_class[$j$])]=1
23:     auc_result= AUC_calculation($\boldsymbol{X}$[,$\mathcal{A}$], tmp_label)
24:     if auc_result<0.5 then
25:         AUC_sign [j]=1
26:     end if
27: end for
15: return $\mathcal{A}$, max_auc, AUC_sign
16: end
```

The algorithm for node threshold selection is presented below.

Algorithm 2 Node_Threshold_Selection

Input(s): $\boldsymbol{X}$, the matrix of training data; $\boldsymbol{\omega}$, the corresponding label vector; $\mathcal{A}$, column indice of attribute with the highest total AUC; *AUC_sign*, the vector of signal to tell whether AUC for this class is smaller than 0.5

Output(s): *thre_result*, the threshold of for this split

```
1:  thre_result=0
2:  uniq_class=unique($\boldsymbol{\omega}$)
3:  tp_array=rep(0,length(uniq_class))
4:  fp_array=rep(0,length(uniq_class))
5:  uniq_splits=sort(unique($\boldsymbol{X}[,\mathcal{A}]$))
6:  total_true=rep(0,length(uniq_class))
7:  for $i$ in 1:length(uniq_class)
8:      total_true[i]=length(which($\boldsymbol{\omega}$ ==uniq_class[i]))
9:  end for
```

```
10: total_false=rep(nrow($\boldsymbol{X}$),length(uniq_class)) – total_true
11: max_harmean=0
12: for $i$ in 1:length(uniq_splits)
13:     indice=which($\boldsymbol{X}[,\mathcal{A}]$<uniq_splits[i])
14:     for $j$ in 1:length(uniq_class)
15:         if AUC_sign[j]==0 then
16:             tp_array[j] = length(which($\boldsymbol{\omega}$[indice]==uniq_class[j]))/total_true[j]
17:             fp_array[j] = length(which($\boldsymbol{\omega}$[indice]!=uniq_class[j]))/total_false[j]
18:         else
19:             tp_array[j] = length(which($\boldsymbol{\omega}$[indice]!=uniq_class[j]))/total_true[j]
20:             fp_array[j] = length(which($\boldsymbol{\omega}$[indice]==uniq_class[j]))/total_false[j]
21:         end if
22:     end for
23:     tmp_harmean=Harmonic_mean(c(tp_array,rep(1,length(uniq_class))-fp_array))
24:     if tmp_harmean > max_harmean then
25:         max_harmean = tmp_harmean
26:         thre_result=uniq_splits[i]
27:     end if
28: end for
29: return thre_result
30: end
```

Then we can have the algorithm for Multi-Class ROC Tree.

Algorithm 3 Multi_Class_ROC_Tree

```
Input(s): $\boldsymbol{X}$, the matrix of training data; $\boldsymbol{\omega}$, the corresponding label vector; $\boldsymbol{\mathcal{A}}$, the set of column
          indices in $\boldsymbol{X}$ as available attributes; cur_id, the current id of this node; max_depth, the
          maximum depth for this tree; min_leaf, the minimum number of observations on each
          node; train_ratio, the ratio of data to be used for training; $\boldsymbol{N_a}$, the number of attribute
          allowed in each node
Output(s): roc_tree, the data frame of the multi-class ROC tree
1: roc_tree=data.frame(id=cur_id, split_var="", thre="",lchild=0,rchild=0,nodelabel= -1,
        nodescore="",oob_score="")
2: Randomly generate the training indices and testing indices of the rows of $\boldsymbol{X}$ based on the
train_ratio
3: Calculate the nodesocre and nodelabel based on the training data $\boldsymbol{X}$[training_indice,]
4: Calculate the oob_score based on the testing data $\boldsymbol{X}$[testing_indice,] for comparison
5: Check whether the stopping criteria is met or not. If it's met, return roc_tree
6: Sample $\boldsymbol{N_a}$ features from $\boldsymbol{\mathcal{A}}$ and set them to be $\boldsymbol{\mathcal{A}'}$
7: Find the attribute $\mathcal{A}$ using training data $\boldsymbol{X}$[training_indice,], the attributes set $\boldsymbol{\mathcal{A}'}$, the label
vector $\boldsymbol{\omega}$[training_indice] and function Node_Attribute_Selection (Algorithm 1)
8: Find the threshold thre_result using the attribute $\mathcal{A}$, training data $\boldsymbol{X}$[training_indice,] and
corresponding label vector $\boldsymbol{\omega}$[training_indice] and function Node_Threshold_Selection
(Algorithm 2)
9: Build the left child by calling Multi_Class_ROC_Tree (Algorithm 3) recursively. Let the left
child id to be cur_id*2, left child max depth = max_depth – 1. Append the return dataframe with
the data frame roc_tree
10: Build the right child based using same function. Let the left child id to be cur_id*2+1, right
child max depth = max_depth – 1. Append the return dataframe with the data frame roc_tree
11: return roc_tree
12: end
```

3.3 Multiclass ROC Random Forest Algorithm

The random forest algorithm [10] was modified and applied towards the multiclass ROC tree. The original random forest algorithm proposed by Leo Brieman was an ensemble framework to combine weak learners. For each individual tree building process in random forest framework, it introduced two kinds of randomness. At each node building stage, it randomly selectes part of the available attributes as possible candidates for the node attribute. Then for each tree, only part of the observations is selected to build the tree. The introduction of these two types of randomness ensures that every tree in the random forest will not be identical. After a certain number of trees are built, the random forest use a bagging method to combine the trees, which will assign each tree equal weight and ask them to vote for the class. The class with the majority votes will be chosen as the prediction result of the random forest. The multiclass ROC random forest apply the framework of random forest as follows.

Algorithm 4 Multi_Class_ROC_Random_Forest

```
Input(s): $\boldsymbol{X}$, the matrix of training examples; $\boldsymbol{\omega}$, the corresponding label vector; $\boldsymbol{N_t}$, the number of
          trees to be generated; $\boldsymbol{N_a}$, the number of attributed needed for each node
Output(s): $\boldsymbol{\mathcal{F}}$, the final forest
1:  set $\boldsymbol{\mathcal{F}}$ to NULL
2:  for $i = 1$ to $\boldsymbol{N_t}$
3:      sample a set of row indices from the original data with replacement noted as
bagging_indice
4:      train $tree_i$ by this sampled data $\boldsymbol{X}$[bagging_indice,], $\boldsymbol{\omega}$[bagging_indice,] with
Multi_Class_ROC_Tree (Algorithm 3)
5:      append $tree_i$ to $\boldsymbol{\mathcal{F}}$
6:  end for
7:  return $\boldsymbol{\mathcal{F}}$
8:  end
```

However, if we directly apply the bagging method on imbalanced data sets, the final result will tend to assign every observations to the majority class, since the class score of each leaf is based on the percentage of each class. Therefore we introduce the prior percentage, which help balance the leaf scores. For each observation, we sum up the scores for each class from every tree in the forest, and then divided them by the prior percentage of each class to balance the class score. The class with the largest balanced score will be the class for this leaf. The detail prediction algorithm is presented below.

Algorithm 5 ROC_Forest_Prediction

```
Input(s):  $\boldsymbol{\mathcal{F}}$, the ROC forest; $\boldsymbol{x}$, a new observation; pred_type, the type of prediction needed;
           prior_prob, the prior distribution of each class
Output(s):  $\widehat{\omega}$, the predicted label for $\boldsymbol{x}$ or $\hat{p}$, the predicted probability for $\boldsymbol{x}$
1:  set score as an empty data frame
2:  for each $tree_i$ in $\boldsymbol{\mathcal{F}}$:
3:      score[$i$,]=ROC_Tree_Prediction($tree_i$, $\boldsymbol{x}$ ,pred_type,id=1)
4:  end for
5:  sum up score for each class and calculate the score sum percentage for each class as $\hat{p}$
6:  $\hat{p}$=$\hat{p}$/prior_prob/sum($\hat{p}$/prior_prob)
7:  If pred_type=="score" then
8:      return $\hat{p}$
9:  else
10:     $\widehat{\omega}$=which.max($\hat{p}$)
11:     return $\widehat{\omega}$
12: end if
13: end
```

4. Performance Evaluation

4.1 Multiclass Classification Evaluation Methods

The performance of imbalanced binary classification can be evaluated by the ROC curve and AUC. However the original ROC curve only works in binary classification problems, so new measures needs to be defined to evaluate the performance of multi-class classification.

4.1.1 Generalized Form of Notations

In [11], these measures for l classes classification problem are defined as following.

- Average Accuracy: $\frac{\sum_{i=1}^{l}\frac{tp_i+tn_i}{tp_i+tn_i+fp_i+fn_i}}{l}$ which measures the average accuracy of each class
- Precision$_\mu$: $\frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l} tp_i+fp_i}$ which measures the micro-average accuracy for positive predictions
- Recall$_\mu$: $\frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l} tp_i+fn_i}$ which measures the micro-average true positive rate
- Precision$_M$: $\frac{\sum_{i=1}^{l}\frac{tp_i}{tp_i+fp_i}}{l}$ which measures the macro-average accuracy for positive predictions
- Recall$_M$: $\frac{\sum_{i=1}^{l}\frac{tp_i}{tp_i+fn_i}}{l}$ which measures the macro-average true positive rate

Note that in multi-class classification, the Precision$_\mu$ and Recall$_\mu$ will be the same since $\sum_{i=1}^{l} tp_i + fp_i$ is the sum of predicted positive for each class, which is the number of observations, and $\sum_{i=1}^{l} tp_i + fn_i$ is the sum of real positive for each class, which is also the number of observations.

4.1.2 Volume Under Surface (VUS)

The VUS is an extension of the AUC. In binary classification, the ROC curve shows the performance of a classifier on a plot with y axis as TPR and x axis as FPR for class 1. And the FPR for class 1 can also be regarded as 1 – TPR for class 0. Therefore the axis in the ROC curve plot can be regarded as TPR for different class.

Subsequently, in multi-class classification problems, a coordinate system similar to ROC curve can be built to measure the performance of classifiers [12]. In an L class classification problem, the dimension of this system will be L and the axis is the TPR for each class. And the classifier will be a surface in this space. The ROC curve in 2D is a degenerate form of this surface. Hence similar to AUC, the Volume Under Surface can be defined to evaluate the performance of a classifier, which involves the calculation of the volume of a convex hull.

4.2 Comparison with SMOTE Algorithms on UCI Repository Data

In this section we compare the performance of ROC tree/random forest with the classical SMOTE algorithm [1] combined with random forest and Ferri's ROC Tree [4] on the following UCI repository data shown in Table 1.

Table 1: Chosen binary classification UCI repository data set

Name	Observation Number	Feature Number	Minority Class Percentage
Letter Recognition A	20000	16	3.95
Optical Recognition of handwritten digits 0	5620	64	9.86
Pen-based Recognition of handwritten digits 0	10992	16	9.4
Ionosphere	351	34	35.9

We ran 10 fold cross validation 10 times on the data and got the following performance. The ROC random forest and SMOTE random forest both contain 100 trees so that their performance is comparable.

Table 2: Algorithm Performance on UCI repository data set

Data Set	Letter A		Opt Digit 0		Pen Digit 0		Ionosphere	
	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC	Accuracy	AUC
Ferri's Gain Ratio	99.5±0.2	98.9±1.4	98.9±1.8	94.2±1.4	99.6±0.3	99.6±0.5	92.0±4.7	90.4±7.0
Ferri's AUC Split	99.5±0.1	99.3±0.7	99.5±0.3	98.5±1.8	99.6±0.2	99.4±0.6	89.6±5.0	89.7±6.7
Proposed ROC Random Forest	99.4±0.1	99.9±0.02	99.6±0.2	99.9±0.03	98.9±0.3	99.8±0.1	91.0±4.3	96.7±3.1
SMOTE Random Forest	99.7±0.1	99.9±0.02	99.7±0.2	99.9±0.04	99.8±0.1	99.9±0.1	90.3±5.0	96.9±2.9

From this table we can see that the performance of ROC random forest is at the same level with the SMOTE random forest when measured by AUC, and both of which outperformed Ferri's method [4] in terms of AUC.

Compared to the SMOTE algorithm, the ROC random forest is much faster since it does not require any pre-processing steps on the data. Let the *n* be the number of training observations at one node, and *k* the number of unique classes, *m* the number of features, and *s* the average number of possible splits along each feature. Then the time complexity of building a ROC tree node is O(m(nlog(n)+kn)) [13], while the time complexity of building a CART tree node is O(m(nlog(n)+sn)). In most numerical features, the number of possible splits s is close to the number of training observations, so the ROC tree node building is faster than CART theoretically, as the total running time depends on each single splits.

4.3 Comparison with Traditional Algorithms on Simulated Data

In this section we compare the performance of multiclass ROC tree/random forest with traditional algorithms on imbalanced data sets, and we can see that this algorithm we proposed works well on imbalanced data sets. There are many situations in multi-class classification, and here we compared the performance of Multi-Class ROC Tree, Multi-Class ROC Random Forest, CART, random forest, support vector machine and random guess on 2 situations. The number of classes is set to be 4, with 1 dominate class of 4750 observations (95% of the total data), and 2 minor classes of 100 observations each (2% of the total data), and 1 rare class of 50 observations (1% of the total data). The random guess is generated by a multi-nominal distribution with probability as the class distribution. The CART is fit using the **rpart** package in R, the random forest is fit using the **randomForest** package in R, and the support vector machine is fit using the **e1071** package in R. All the performance is measured by the out of bag testing data, which is generated using the same method as the training data.

4.2.1 Setting 1: 2 dimensions

In this setting, we generate observations from Gaussian distribution with 2 dimensions so that it is easy to visualize the data. All the variables have a variance of 1. Class 1 is centered at (0, 0), Class 2 at (2, 2), Class 3 at (-2, -2), Class 4 at (-2, 2).

The performance of Multi-Class ROC Tree, CART and random guess is showed in Table 3. The ROC tree has slight advantage over CART in Recall_M and VUS.

Table 3: Performance of Classifiers on Setting 1

	Average Accuracy	Precision_μ	Precision_M	Recall_M	VUS
ROC Tree	0.9809	0.9618	0.7345	0.5794	0.09657
CART	0.9817	0.9634	0.8110	0.5360	0.08933
Random Guess	0.9513	0.9026	0.2476	0.2473	0.04167

4.2.2 Setting 2: 10 dimensions

In this setting, high dimension data is used to check the performance of multi-class ROC Tree. All the data are normally generated with variance 1 and the centers for each class is listed below.

- Class 1: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
- Class 2: (1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
- Class 3: (-1, -1, -1, -1, -1, -1, -1, -1, -1, -1)
- Class 4: (-1, 1, -1, 1, -1, 1, -1, 1, -1, 1)

The following table 4 shows the performance of each classifier on this setting. CART didn't predict any observation to class 4 so the Precision_M is NA. The Multi-Class ROC Tree outperformed both CART and random forest with 10 trees. The ROC random forest has large advantage over other classifiers.

Table 4: Performance of Classifiers on Setting 2

	Average Accuracy	Precision_{μ}	Precision_{M}	Recall_{M}	VUS
ROC Tree	0.9727	0.9454	0.5535	0.4797	0.07995
CART	0.9733	0.9466	NA	0.2907	0.04845
Random Guess	0.9513	0.9026	0.2476	0.2473	0.04167
ROC RF (100 trees)	0.6943	0.3886	0.3031	0.8121	0.1353
RF (100 trees)	0.9794	0.9588	0.9896	0.38	0.0633
SVM	0.9857	0.9714	0.9232	0.6191	0.1032

The table 5 shows the sensitivity and specificity of each class in detail for random forest, SVM and ROC random forest.

Table 5: Sensitivity and Specificity of Classifiers on Setting 2

Algorithm	Random Forest		SVM		ROC Random Forest	
Class	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity
1	0.9993684	0.192	0.9966316	0.492	0.3587368	0.984
2	0.24	0.9997959	0.48	0.9983673	0.95	0.757551
3	0.15	0.9995918	0.5	0.9983673	0.94	0.7991837
4	0.18	1	0.5	1	1	0.8220202

We can see that the ROC random forest actually puts equal weight to each class, and it tries to maximize the sensitivity and specificity for each class, while SVM and traditional random forest focus more on class 1 because it has the largest number of observations.

4.4 Summary

In this paper we propose a new tree based method to deal with the imbalanced data classification problem that does not require any pre-processing steps. This method performs as well as the SMOTE random forest on binary classification problems when using the same parameters and measured by AUC, while it is much faster than SMOTE random forest since it does not require any pre-processing steps. This algorithm also works on multi-class classification problems, where it puts equal emphasis on the sensitivity and specificity of each class, big and small alike. We can further adjust the emphasis by fine-tuning the prior probability parameter in ROC random forest.

References

[1] Chawla, Nitesh V., et al. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research (2002): 321-357.

[2] Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on. Vol. 1. IEEE, 2001.

[3] Elkan, Charles. "The foundations of cost-sensitive learning." International joint conference on artificial intelligence. Vol. 17. No. 1. LAWRENCE ERLBAUM ASSOCIATES LTD, 2001.

[4] Ferri, César, Peter Flach, and José Hernández-Orallo. "Learning decision trees using the area under the ROC curve." ICML. Vol. 2. 2002.

[5] Hossain, M. Maruf, Md Rafiul Hassan, and James Bailey. "ROC-tree: A Novel Decision Tree Induction Algorithm Based on Receiver Operating Characteristics to Classify Gene Expression Data." SDM. 2008.

[6] Powers, David Martin. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation." (2011).

[7] Bishop, Christopher M. "Pattern Recognition." Machine Learning (2006).

[8] Landgrebe, Thomas, and R. Duin. "A simplified extension of the area under the ROC to the multiclass domain." In Seventeenth annual symposium of the pattern recognition association of South Africa, pp. 241-245. 2006.

[9] Breiman, Leo, et al. Classification and regression trees. CRC press, 1984.

[10] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.

[11] Sokolova, Marina, and Guy Lapalme. "A systematic analysis of performance measures for classification tasks." Information Processing & Management 45, no. 4 (2009): 427-437.

[12] Landgrebe, Thomas, and R. Duin. "A simplified extension of the area under the ROC to the multiclass domain." In Seventeenth annual symposium of the pattern recognition association of South Africa, pp. 241-245. 2006.

[13] Fawcett, Tom. "An introduction to ROC analysis." Pattern recognition letters 27.8 (2006): 861-874.

[14] Galar, Mikel, et al. "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches." IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42.4 (2012): 463-484.

[15] Allwein, Erin L., Robert E. Schapire, and Yoram Singer. "Reducing multiclass to binary: A unifying approach for margin classifiers." Journal of machine learning research 1, no. Dec (2000): 113-141.