Masterthesis:
Random Forest for Prediction with Unbalanced Data

Masterthesis im Studiengang

„Stochastic Engineering in Business and Finance“

vorgelegt von:

Name: Greg Peargin

Matrikelnr: 08435816

eingereicht bei Prof. Dr. Manfred Gruber, Fakultät Informatik und Mathematik

Erklärung

Hiermit erkläre ich, dass ich die Arbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

München, 11.02.19 _____

Unterschrift

Table of Contents

1. Introduction

The intent of this paper is to further explore the methodology applied to the class imbalance problem posed in M. Zięba et al. (2014) [68]. In the original paper, a boosted Support Vector Machine ensemble was created and decision rules were extracted using a JRIP algorithm, thus sacrificing some accuracy for a more interpretable model. Our goal is to determine whether, at the cost of interpretability, a higher degree of accuracy can be achieved using Random Forests, therefore we will compare our results to those of the boosted SVM ensemble itself. In the process, we will explore different strategies for tackling class imbalance and ultimately determine which of these achieves the best results in the context of our dataset.

The data, retrieved from the UCI Machine Learning Repository [23], were originally obtained from the Department of Thoracic Surgery of the Medical University of Wroclaw and pertain to whether patients who had undergone major lung resections for primary lung cancer survived a one-year period following their surgery. While originally containing data for over 1,200 patients, the final dataset utilized by the authors, after removing observations with missing values, contains information on only 470 patients. Of these, 400 survived the one-year period, while the other 70 died. Additionally, while the original dataset contained 139 pre-, peri-, and post-operative predictors, the final dataset contains only pre-operative predictors, as the authors were interested in providing doctors with a means of assessing the risk of performing surgery on a given patient before the surgery is carried out. Of 36 original pre-operative predictors, 16 were chosen for the final dataset by performing feature selection with an information gain criterion [63]. Of these, three predictors are numeric, three are categorical, and ten are binary. Descriptions of the variables are given in Table 1.

Table 1: Variable descriptions for thoracic surgery dataset

ID	Description	InfoGain
PRE14	*T in clinical TNM (size of the original tumor, from OC11 (smallest) to OC14 (largest))*	*0.029*
DGN	*Diagnosis (specific combination of ICD-10 codes for primary and secondary as well as multiple tumors if any)*	*0.013*
PRE4	*Forced vital capacity (FVC)*	*0.008*
PRE7	*Pain (pre-surgery)*	*0.008*
AGE	*Age at surgery*	*0.008*
PRE6	*Performance status (Zubrod scale)*	*0.007*
PRE11	*Weakness (pre-surgery)*	*0.004*
PRE9	*Dyspnoea (pre-surgery)*	*0.004*
PRE10	*Cough (pre-surgery)*	*0.003*
PRE8	*Haemoptysis (pre-surgery)*	*0.003*
PRE25	*PAD (peripheral arterial diseases)*	*0.003*
PRE19	*MI up to 6 months*	*0.003*
PRE5	*Volume that has been exhaled at the end of the first second of forced expiration (FEV1)*	*0.002*
PRE32	*Asthma*	*0.002*
PRE30	*Smoking*	*0.002*
PRE17	*Type 2 DM (diabetes mellitus)*	*0.002*
Risk1Y	*1-year survival period (True value if patient died)*	

The remainder of this paper will be organized as follows: first, we will review the methodologies of Decision Trees and Random Forests; next, we will discuss the difficulties of class imbalance and the different strategies we will employ to tackle them; and finally, we will summarize the results of our employed methods and compare them with those of the original paper.

2. Decision Trees

In order to understand Random Forests, we must first familiarize ourselves with Decision Trees. Decision Tree Classifiers work by recursively partitioning the feature space of the data through a series of logical questions, which split the data into disjunct sets of decreasing size until some stopping rule is fulfilled or until each terminal "node" of the tree contains only one observation. These questions take the following forms:

- Is $x \leq c$?, c constant (for quantitative variables)
- Is $x \in A$?, A category (for categorical variables)

Depending on whether or not a data point fulfills this criterion, it will be sent to a left (in the case of a "yes") or right (in the case of a "no") "daughter" node. At each split, the set of all such questions across all variables is considered, and the question which creates the split with the greatest decrease in "node impurity" is chosen. Node impurity is typically defined either by the Gini Index for CART models [13] or by information gain for ID3, C4.5, and C5.0 models [46, 47]. If a tree is allowed to be split until all terminal nodes are "pure," i.e. until they contain only observations from one class, overfitting will occur. To avoid this, stopping rules can be introduced to the model. Possible examples of stopping rules are minimum terminal node sizes, maximum tree lengths, and minimum impurity decreases arising from a split. Alternatively, a tree can be grown to maximum size and be "pruned" backwards, as outlined in [13]. Figure 1 shows an example tree for the thoracic surgery dataset using only two numeric variables, "PRE4" and "AGE," and the resulting partition of the feature space. Note that some splits resulted in a prediction of "FALSE" for both daughter nodes. This is because "TRUE" did not occur as frequently in either of the two daughter nodes after the split, and the predicted class for a terminal node is the most frequently occurring class in that node based on the training data.

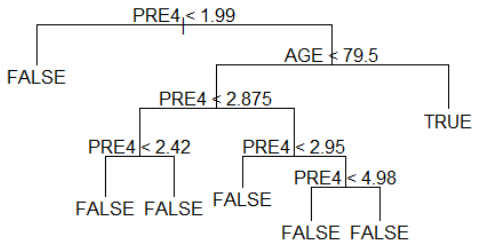

Figure 1a: Example tree for thoracic surgery dataset

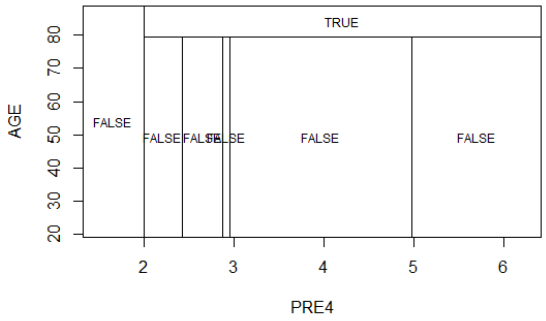

Figure 1b: Partition of feature space for example tree

Decision Trees are particularly well suited for handling categorical data, as the nature of the splitting algorithm eliminates the need to convert categorical attributes into binary variables. Continuous variables are treated as if they were countable, as the question "is $x \leq c$?" has only as many possible values for c as there are unique values for the attribute.

3. Random Forests

Decision Trees are susceptible to the bias-variance tradeoff [30]: a tree of maximum size will be unbiased, but will not be robust to noise due to overfitting. Similarly, a smaller tree may generalize better, but will consequently be biased due to impure terminal nodes. Random Forests [12] present a solution to this problem through the creation of an ensemble of decorrelated trees. Low correlation between individual trees is guaranteed in two ways: first, only a random subset of features is considered for possible splits at each non-terminal node; and second, each tree is grown using a bootstrap sample from the training set as large as the training set itself, a method known as "bootstrap aggregating" or "bagging" [10]. This provides Random Forests with a built-in method for estimating the generalization error, known as the "out-of-bag" estimate [11] (for conventional reasons, however, we will use ten-fold cross-validation [52] to estimate the errors of our models for the thoracic surgery dataset). Trees in the ensemble are grown to maximum size and are not pruned. The ensemble is grown to contain a large number of trees which, in the final model, each cast a vote for the class of a "test" data point. The final predicted class is the class with the most votes. The benefit of using a Random Forest model, as opposed to a Decision Tree, is that it is more robust to noise while maintaining low bias.

4. Class Imbalance

In this section, we will clarify what class imbalance is, the issues that it causes in classification problems, and why class imbalance causes these issues.

Table 2: Confusion matrix for Random Forest on thoracic surgery dataset

		Predicted		**Class Error**
		FALSE	TRUE	
Actual	FALSE	40	0	0
	TRUE	7	0	1

Class imbalance exists in any dataset where the prior probabilities of its classes differ. According to this definition, nearly any dataset with labeled data will possess some degree of class imbalance. However, when referring to a class imbalance problem, we typically mean a situation where a dataset contains many more negative (majority) examples than positive (minority) examples. A measure of the degree of imbalance present in a dataset is the Imbalance Ratio [45], defined as the number of negative examples divided by the number of positive examples. In the thoracic surgery dataset, we have 400 "FALSE" examples and 70 "TRUE" examples; thus, our Imbalance Ratio is 5.71.

The problem that class imbalance poses is that most algorithms are poor at correctly classifying positive examples, often with accuracies of 0% [25], yet it is often the case that we are more concerned with accurately predicting these minority class examples. Without any strategy for addressing this issue, models will often be trivial, as they are nearly or completely incapable of distinguishing between the two classes. To see this, consider the confusion matrix for a Random Forest model trained with 90% of the thoracic surgery data in Table 2. The model simply predicts all of the test examples as "FALSE," giving an error rate of 14.9%. In fact, the more imbalance present in a dataset, the lower the estimated error will be. As a result, normal measures of accuracy and error are deceptive when dealing with imbalanced data, therefore we will need to use alternative performance measures in order to judge the quality of our models.

4.1 Performance Measures

A number of performance measures exist for models on imbalanced datasets. The following are among the most popular:

- $precision = \frac{true\ positives}{predicted\ positives}$
- $recall = \frac{true\ positives}{total\ positives}$
- $F_1 - measure = 2\frac{precision \cdot recall}{precision + recall}$
- $Gmean = \sqrt{sensitivity \cdot specificity}$

Note that the sensitivity and specificity are the same as the true positive rate and true negative rate, respectively. While it is conventional under ordinary circumstances to report model performance in terms of error, it should be noted that the above measures are all accuracy measures and should be maximized.

A fifth performance measure is the receiver operating characteristic (ROC) curve [24]. The ROC curve plots the false positive rate of a model on the x-axis against the true positive rate on the y-axis. The different values along the curve are obtained by allowing the threshold for a positive classification to vary. In the standard Random Forest model, we classify an observation as positive when there are more votes for the minority class than for the majority class; in other words, the threshold is 50%. An ideal model would be found in the upper left corner of the ROC curve and the area under the curve (AUC) would be equal to 1. It has been argued that a Precision-Recall (PR) curve is preferable to the ROC curve for classification with imbalanced data [51], though both are generally accepted.

4.2 Difficulties of Imbalanced Data

To understand the rationale of the methods for tackling class imbalance, it is important that we understand why class imbalance makes classification of positive examples difficult. The following reasons are given in [25]:

1. *Small sample size*: If positive examples are sufficiently rare, or if the size of the training set is small, there may not be enough information on the minority class for an algorithm to identify the subspace occupied by it.
2. *Class separability*: If our data were perfectly separable, then an algorithm capable of reasonably approximating the decision boundary should have no difficulty distinguishing between the two classes. Thus, the class imbalance problem implies at least a minimal degree of class inseparability. Plots 1-3 of Figure 2 contrast datasets with partial and complete inseparability, as well as low vs. high variance of the minority class in completely inseparable datasets. These datasets all possess the same level of imbalance, with an Imbalance Ratio of 10. As the degree of class inseparability in the data increases from Plot 1 to Plot 3, we see that the cost of greater accuracy for positive examples increases in terms of accuracy for negative examples.
3. *Small disjuncts*: If the subspace occupied by the minority class does not constitute a single region, but rather multiple subregions, then the difficulties of small sample size and class inseparability may be compounded by this fact, as the disjuncts may be inseparable themselves, and will contain only a fraction of the positive examples. High dimensionality in the data often causes small disjuncts. Plot 4 of Figure 2 shows an example of a dataset containing small disjuncts.

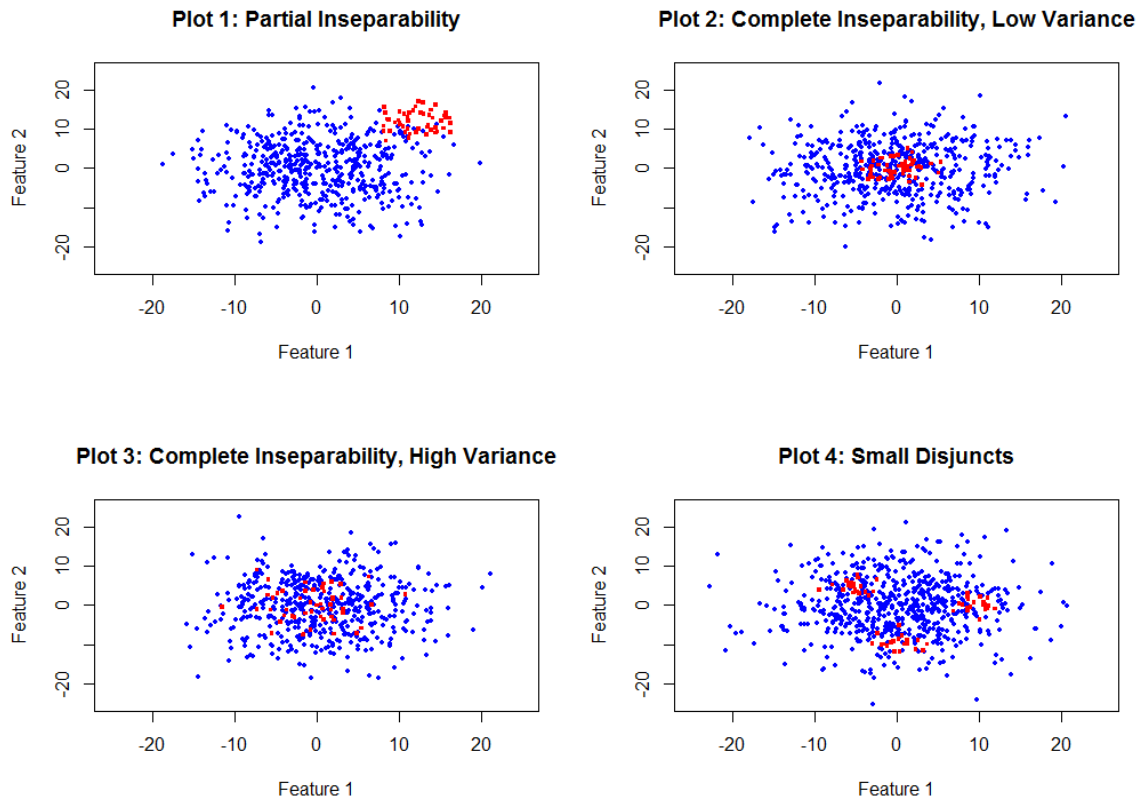

Figure 2: Different types and degrees of class inseparability

5. Methods for Imbalanced Data

In this section, we will discuss the methods we can use to address the class imbalance in our data. These methods can be broken down into four categories: data-level, cost-sensitive, ensemble, and algorithm-level approaches [25].

5.1 Data-Level Preprocessing Methods

Data-level or preprocessing methods address class imbalance by changing the frequencies of observations within either class, thus directly influencing their prior probabilities. This is achieved by means of two methods: oversampling and undersampling. When oversampling, one creates more positive observations by randomly sampling the minority class training examples with replacement, and appending these on to the existing training set. Conversely, undersampling is performed by randomly sampling without replacement from the negative examples of the training set, and the entirety of the majority examples are replaced with this new, smaller training set. Both methods increase the prior probability of the minority class and decrease that of the majority class in the training set.

In practice, both of these methods are flawed, but they constitute the foundational ideas behind all other preprocessing methods for class imbalance. Undersampling can result in a loss of useful information, while oversampling can overfit the model to individual positive examples. The remaining methods in this section

will seek to modify the oversampling and undersampling procedures in order to address their fundamental issues.

5.1.1 One-Sided Selection

One-sided selection offers a solution to the problem of information loss through undersampling by proposing that certain negative examples are more informative for classification than others [36]. Majority class samples are sorted into four categories: (1) samples suffering from class-label noise; (2) borderline examples which can be easily misclassified due to attribute noise; (3) redundant examples which are not at risk of being misclassified, but which increase classification costs; and (4) "safe" examples which should be kept in the training set. The goal of one-sided selection is to remove all borderline, noisy, and redundant samples from the majority class, leaving only the safe examples. Redundant examples can be identified by classifying each sample in the training set according to the 1-NN (first nearest neighbor) rule [22], using only the positive examples and one randomly selected negative example for comparison. Meanwhile, borderline and noisy examples can be identified with the help of Tomek links [59], defined as follows:

> Denote $\delta(x, y)$ as the distance between x and y. The pair (x, y) is called a Tomek link if no example z exists such that $\delta(x, z) < \delta(x, y)$ or $\delta(y, z) < \delta(y, x)$.

The procedure for one-sided selection is outlined in Algorithm 1.

Algorithm 1 One-Sided Selection

1. Let S be the original training set.
2. Initially, C contains all positive examples from S and one randomly selected negative example.
3. Classify S with the 1-NN rule using the examples in C, and compare the assigned concept labels with the original ones. Move all misclassified examples into C that is now consistent with S while being smaller.
4. Remove from C all negative examples participating in Tomek links. This removes those negative examples that are believed borderline and/or noisy. All positive examples are retained. The resulting set is referred to as T.

5.1.1.1 A Note on Distance Measures

Because our data contain both numeric and categorical features, proper scaling between continuous and nominal predictors must occur to ensure that neither type of variable has greater influence on the calculated distance. One suitable distance measure is the Heterogeneous Value Distance Metric (HVDM) [61], defined as follows:

$$HVDM(x, y) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)}$$

where x and y are input vectors, $d_a(x, y)$ is the difference between x and y for attribute a, and m is the number of attributes. $d_a(x, y)$ takes on the following values:

$$d_a(x, y) = \begin{cases} 1, \text{if } x \text{ or } y \text{ is unknown; otherwise ...} \\ normalized\_vdm_a, \text{if } a \text{ is nominal} \\ normalized\_diff_a, \text{if } a \text{ is continuous} \end{cases}.$$

Furthermore,

$$normalized\_vdm_a(x,y) = \sqrt{\sum_{c=1}^{C}\left|\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}}\right|^2},$$

$$normalized\_diff_a(x,y) = \frac{|x-y|}{4\sigma_a},$$

where $N_{a,x}$ is the number of instances in the training set that have value x for attribute a, $N_{a,x,c}$ is the number of instances in $N_{a,x}$ that have output class c, and C is the number of classes. Note that values of $normalized\_vdm_a$ are constrained between zero and $\sqrt{C}$ and, assuming differences of continuous variables are normally distributed, dividing by four standard deviations gives $normalized\_diff_a$ a range of approximately one. However, experiments in [61] found very little difference between the average distances of $normalized\_vdm_a$ and $normalized\_diff_a$ even for multi-class problems with many output levels.

5.1.2 SMOTE

The Synthetic Minority Oversampling Technique (SMOTE) [16] provides a solution to the issue of overfitting individual positive examples when performing oversampling. Instead of using random sampling with replacement, new data is synthesized by linearly interpolating between minority class samples. For a chosen value of k, a single neighbor of the k nearest positive neighbors of a positive example is selected at random. The differences between the attributes of these two examples are multiplied by a random value between zero and one, and are then added to the values of the attributes of the original example. Thus, a new example is synthesized at a random location on a line between the original example and one of its nearest minority class neighbors. This allows a model to identify broader classification regions for the minority class, instead of individual examples.

The amount of data synthesized is determined by an "oversample percentage" N. For values of N less than 100, N% of the size of the minority class training set T (rounded to the nearest integer) new examples will be synthesized. Once the number of examples to be synthesized is known, a sample of the same size is drawn from T without replacement, and one new example is synthesized for each point in the sample. For values of N greater than or equal to 100, N signifies the number of new examples to be synthesized for each example in the training set. Thus, beyond a value of 100, N must be a multiple of 100.

Algorithms 2 and 3 give the pseudocode for the SMOTE algorithm and the function for generating synthetic examples, respectively. Note that a method for synthesizing data with nominal attributes is not detailed. In [16], the authors considered separate methods for data with nominal or mixed features, SMOTE-N and SMOTE-NC. These methods propose choosing the value of a categorical attribute of a synthesized example to be the mode of its k nearest positive neighbors. In many software applications, however, the value is simply randomly chosen to be either the value of the original point or that of the selected nearest neighbor.

5.1.3 SMOTE-Tomek

Oversampling and undersampling methods can also be combined in order to mitigate the issues associated with either approach. SMOTE-Tomek [4] is one such hybrid technique, combining synthetic data generation with one-sided selection. Because the number of Tomek links in a data set depends on the number

Algorithm 2 SMOTE

1. **function** $SMOTE(T, N, k)$
   Input: Number of minority class samples T; Amount of SMOTE N%; Number of nearest neighbors k
   Output: $(N/100) * T$ synthetic minority class samples
   Variables: $Sample[][]$: array for original minority class samples; $newindex$: keeps a count of number of synthetic samples generated, initialized to 0; $Synthetic[][]$: array for synthetic samples
2. **if** $N < 100$ **then**
3. Randomize the T minority class samples
4. $T = N/100 * T$
5. $N = 100$
6. **end if**
7. $N = (int)N/100$ #The amount of SMOTE is assumed to be in integral multiples of 100.
8. **for** $i = 1$ to T **do**
9. Compute KNN for i, and save the indices in the $nnarray$
10. $POPULATE(N, i, nnarray)$
11. **end for**
12. **end function**

Algorithm 3 Function to generate synthetic samples

1. **function** $POPULATE(N, i, nnarray)$
   Input: Instances to create N, Original sample index i, Array of nearest neighbors $nnarray$
   Output: N new synthetic samples in $Synthetic$ array
2. **while** $N \neq 0$ **do**
3. $nn = random(1, k)$
4. **for** $attr = 1$ to $numattrs$ **do** $\#numattrs$ = Number of attributes
5. Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
6. Compute: $gap = random(0,1)$
7. $Synthetic[newindex][attr] = Sample[i][attr] + gap \cdot dif$
8. **end for**
9. $newindex++$
10. $N = N - 1$
11. **end while**
12. **end function**

of examples in the minority class, the SMOTE component of this technique is carried out first.

5.1.4 Other Data-Level Preprocessing Methods

Many other extensions of oversampling, undersampling, and hybrid resampling approaches have been developed. Other undersampling techniques include the Condensed Nearest Neighbor Rule (US-CNN) [29], the Neighborhood Clearing Rule (NCL) [38], Class Purity Maximization (CPM) [66], Undersampling Based on Clustering (SBC) [64, 65], and NearMiss approaches [67], among more advanced techniques involving evolutionary algorithms, ensembles, or clustering [25]. Most oversampling methods are extensions of SMOTE, for example: Borderline-SMOTE [28], Adjusting the Direction of the Synthetic Minority Class Examples (ADOMS) [56], Adaptive Synthetic Sampling Approach (ADASYN) [31], Random

Oversampling Examples (ROSE) [43], Safe-Level-SMOTE [14], Density-Based SMOTE (DBSMOTE) [15], Majority Weighted Minority Oversampling Technique (MWMOTE) [3], and Mahalanobis Distance-Based Oversampling Technique (MDO) [1]. Additional hybrid methods include SMOTE with Edited Nearest Neighbor Rule (SMOTE-ENN) [5], Agglomerative Hierarchical Clustering (AHC) [21], SPIDER [7, 55], SMOTE-RSB [48], and SMOTE-IPF [50].

5.2 Cost-Sensitive Learning

Cost-sensitive learning introduces the idea of unequal misclassifications costs for different types of misclassification as a solution to class imbalance. Consider the 0-1 loss function of a typical error-minimizing model: correctly classified observations carry a misclassification cost of zero, while misclassified examples have a misclassification cost of one; minimizing the sum of the misclassification costs of the model, or simply minimizing the number of misclassified observations, is therefore equivalent to minimizing the error [25]. Assuming different costs for different types of misclassification, we can obtain a cost matrix like that illustrated in Table 3.

We can express the expected cost $R(i|x)$ of classifying instance x as belonging to the i-th class as:

$$R(i|x) = \sum_{j=1}^{M} P(j|x) \cdot C(i,j),$$

where $C(i,j)$ is the cost associated with misclassifying an observation belonging to the j-th class as belonging to the i-th class, and $P(j|x)$ is the estimated probability of instance x belonging to the j-th class, with a set of M classes. Predicting an observation as belonging to the class for which the expected cost is lower, we can express the condition for which we will predict an observation as belonging to the minority class as:

Table 3: Cost matrix

	True positive	**True negative**
Predicted positive	$C(0,0)$	$C(0,1)$
Predicted negative	$C(1,0)$	$C(1,1)$

$$P(0|x) \cdot C(1,0) + P(1|x) \cdot C(1,1) \leq$$
$$P(0|x) \cdot C(0,0) + P(1|x) \cdot C(0,1)$$

After collecting like terms and accounting for $C(0,0) = C(1,1) = 0$, this reduces to:

$$P(0|x) \cdot C(1,0) \leq P(1|x) \cdot C(0,1).$$

Finally, we obtain a threshold p^* where we classify an observation x as positive if $P(1|x) \geq p^*$:

$$p^* = \frac{C(1,0)}{C(1,0) - C(0,1)}.$$

Thus, as we increase the cost associated with a false negative misclassification, we bias our model in favor of positive examples by raising the threshold required for classifying an observation as negative.

When using instance weighting for classification trees [58], misclassification costs are converted to weights for individual classes. These weights then impact the impurity decreases when splitting a node, as well as the class ratios in the end nodes. Weighted Random Forests [18] apply this same methodology to their trees.

5.3 Ensemble Methods

Ensemble methods for imbalanced data consist of two kinds of approaches: bagging and boosting. Both kinds of approaches are characterized by training many models, using a "weak learner" as a base classifier, and letting these models vote on the

Algorithm 4 AdaBoost

1. **Input:** Training set $S = \{\boldsymbol{x}_i, y_i\}, i = 1, \dots, N$; and $y_i \in \{-1, +1\}$; T: Number of iterations; I: Weak learner
   Output: Boosted classifier: $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$, where h_t, α_t are the induced classifiers (with $h_t(x) \in \{-1, +1\}$) and their assigned weights, respectively
2. $D_1(i) \leftarrow 1/N$ for $i = 1, \dots, N$
3. **for** $t = 1$ to T **do**
4. $\quad h_t \leftarrow I(S, D_t)$
5. $\quad \varepsilon_t \leftarrow \sum_{i, y_i \neq h_t(x_i)} D_t(i)$
6. $\quad$ **if** $\varepsilon_t > 0.5$ **then**
7. $\quad\quad T \leftarrow t - 1$
8. $\quad\quad$ **return**
9. $\quad$ **end if**
10. $\quad \alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
11. $\quad D_{t+1}(i) = D_t(i) \cdot e^{(-\alpha_t h_t(\boldsymbol{x}_i) y_i)}$ for $i = 1, \dots, N$
12. $\quad$ Normalize D_{t+1} to be a proper distribution
13. **end for**

final prediction of test data. The two approaches differ in how they guarantee the uniqueness of their constituent classifiers (an ensemble of identical classifiers would be meaningless). As noted earlier, Random Forests are a kind of bagged classifier. Bagging [10] works by performing random sampling with replacement to construct the training sets of the individual classifiers, while boosting [53] assigns weights to individual samples in the training set, and updates these after the creation of each new classifier according to whether or not these samples were misclassified when resubstituting them into the most recently created classifier. Updating the weights shifts the decision boundary towards the misclassified observations and away from the correctly classified observations. Some ensembles, such as AdaBoost [26], the representative algorithm for the family of boosting algorithms, assign additional weights to the individual classifiers themselves, and make a final prediction based on a weighted vote. The pseudocode for the AdaBoost algorithm is outlined in Algorithm 4. A Boosted Random Forest [44] is a Random Forest of boosted trees, where the weights are normalized to form a probability distribution to instruct the bagging procedure.

5.3.1 SMOTEBoost and SMOTEBagging

Just as oversampling and undersampling methods can be combined, so too can resampling methods be used in conjunction with ensemble methods. Not to be confused with "First SMOTE then Boost," SMOTEBoost [17] incorporates SMOTE into a boosting algorithm, the procedure for which is detailed in Algorithm 5. Similarly, SMOTEBagging [34, 60] performs SMOTE on the training set before a bag is drawn for each weak learner in the ensemble.

5.3.2 Other Ensemble Methods

Many other bagging, boosting, and hybrid/double ensemble methods have been developed specifically for imbalanced datasets. To name a few: UnderBagging [2], RUSBoost [54], EUSBoost [27], and EasyEnsemble [42].

Algorithm 5 SMOTEBoost

1. Given: Set $S\{(x_1, y_1), \ldots, (x_m, y_m)\}\, x_i \in X$, with labels $y_i \in Y = \{1, \ldots, C\}$, where $C_m, (C_m < C)$ corresponds to a minority class.
2. Let $B = \{(i, y): i = 1, \ldots, m, y \neq y_i\}$
3. Initialize the distribution D_1 over the examples, such that $D_1(i) = 1/m$.
4. For $t = 1, 2, 3, 4, \ldots, T$
5. Modify distribution D_t by creating N synthetic examples from minority class C_m using the SMOTE algorithm.
6. Train a weak learner using distribution D_t
7. Compute weak hypothesis $h_t: X \times Y \rightarrow [0,1]$
8. Compute the pseudo-loss of hypothesis h_t: $\varepsilon_t = \sum_{(i,y)\in B} D_t(i, y)\left(1 - h_t(x_i, y_i) + h_t(x_i, y)\right)$
9. Set $\beta_t = \varepsilon_t/(1 - \varepsilon_t)$ and $w_t = (1/2) \cdot \left(1 - h_t(x_i, y) + h_t(x_i, y_i)\right)$
10. Update D_t: $D_{t+1}(i, y) = (D_t(i, y)/Z_t) \cdot \beta_t^{w_t}$, where Z_t is a normalization constant chosen such that D_{t+1} is a distribution.
11. Output the final hypothesis $h_{fn} = \arg\max_{y\in Y} \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t}\right) \cdot h_t(x, y)$

5.4 Algorithm-Level Methods

Algorithm-level approaches to class imbalance directly modify the learning procedures of classifiers themselves. For Decision Trees, algorithm-level methods have focused primarily on alternative splitting criteria. One such criterium is the Hellinger distance [32], which is a measure of divergence between two probability distributions and is calculated with the Bhattacharyya coefficient [6]. For a two-class problem in countable space, the Hellinger distance can be expressed as:

$$d_H\left(P(m^+), P(m^-)\right) = \sqrt{\sum_i \left(\sqrt{P(Y^+|X_i)} - \sqrt{P(Y^-|X_i)}\right)^2},$$

which can be derived from the confusion matrix as follows:

$$d_H(TPR, FPR) = \sqrt{\left(\sqrt{TPR} - \sqrt{FPR}\right)^2 + \left(\sqrt{1 - TPR} - \sqrt{1 - FPR}\right)^2}.$$

Hellinger distance trees [19, 20] have been shown to perform well in cases of class imbalance. Figure 3 [25] shows a comparison of partitions of a two-dimensional feature space constructed from the "yeast" dataset [23], where multiple classes have been combined to form a majority class such that the data possess an Imbalance Ratio of 41.6. The left and right sides show partitions resulting from Gini and Hellinger trees, respectively, while parts (a) and (b) show complete and localized views of the feature space. Note two major differences between the two partitions: firstly, that the positive classification regions resulting from the Hellinger trees are bounded, reducing unnecessary misclassification of the majority class; and secondly, that their positive classification regions are broader than those of the Gini trees, causing them to be more robust to attribute noise in the minority class.

5.4.1 Other Algorithm-Level Methods

Other proposals for splitting criteria suitable for handling class imbalance include the off-centered entropy [39], minority entropy [8], and Class Confidence Proportion [41].

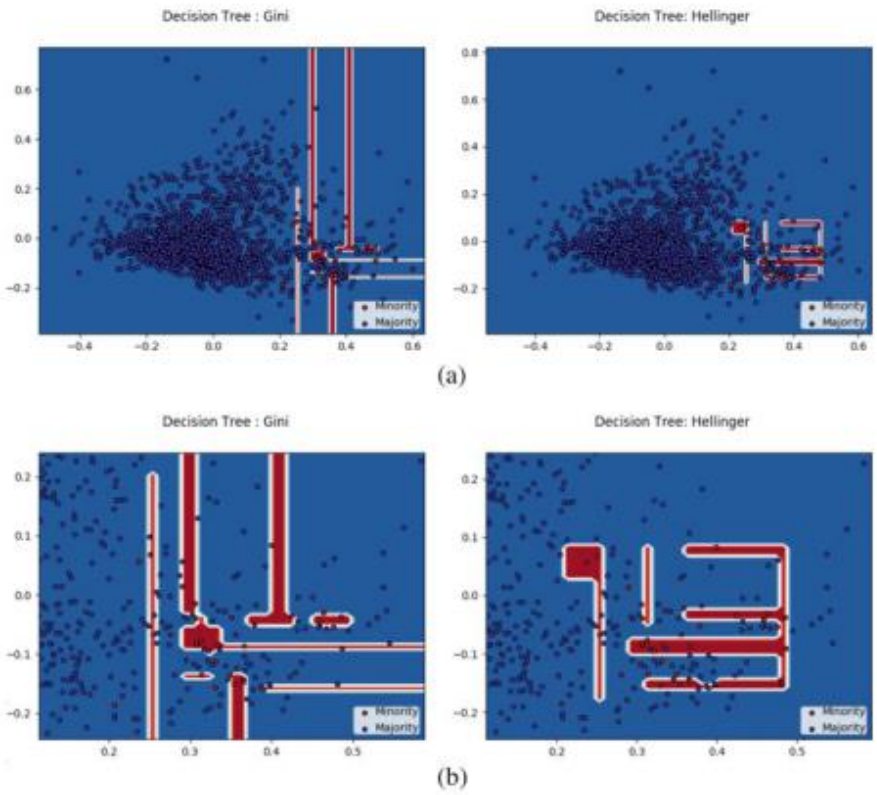

Figure 3: Comparison of Gini and Hellinger trees. Source: Fernandéz, et al., 2018.

6. Applications to the Thoracic Surgery Dataset

In this section, we will state our chosen methods for our applications to the thoracic surgery dataset, then display and discuss the results of our experiments.

6.1 Methodology

Our experiments on the thoracic dataset will take a two-staged approach. In the first stage, we will perform a grid search across a shared set of parameter values and make a side-by-side comparison of the best models obtained for each method. In addition to reporting the parameter values for these models, our metrics for judging each method will be the Gmean (calculated from the 10-fold cross-validated specificity and sensitivity), as well as a benchmark for the computational time of the cross-validated output of the optimal model. Computationally expensive methods will be excluded from a second stage of optimization, where the set of parameter values will be specific to each method and will depend on the optimal values resulting from the first stage. Our applied methods will include: a baseline Random Forest model (**RF**), random oversampling and undersampling (**RUSROS**), one-sided selection with additional random oversampling (**OSS**), SMOTE with random undersampling (**SMOTE**), SMOTE-Tomek (**STMK**), cost-sensitive learning with class weights (**CSL**), SMOTEBagging and SMOTEBoost with random undersampling (**SBAG** and **SBST**, respectively), and a Random Forest constructed from Hellinger trees (**HDRF**). The original Boosted-SVM for Imbalanced Data method from [68] is denoted as **BSI**.

6.1.1 Packages Used

The experiments on the thoracic dataset were carried out in R. Though not an exhaustive list, the following packages represent those which were instrumental to carrying out the aforementioned methods: "caret" [37]; "CORElearn" [49]; "randomForest" [40]; "ranger" [62]; "rpart" [57]; and "UBL" [9]. It should be noted here that, due to the lack of support for random feature selection at individual splits among the available Decision Tree packages in R, alterations were made to the ensemble methods mentioned above. Intead of using Decision Trees, the ensembles generated from the SMOTEBoost method consist of 100 Random Forests. For SMOTEBagging, Decision Trees were kept as the weak learners, and random feature selection was implemented using the Random Subspace Method [33], where each tree is grown using a randomly selected subset of the variables.

6.1.2 Parameter Values

In the first stage of our experiments, parameter values used in our grid searches are shared across all methods (where applicable) to allow for better comparison of their performance. The values searched across possess a wide range and a large step size. This was done in the interest of time, as some of the methods used are quite computationally expensive. It will be noted here that the values of *ntree*, the number of trees in the Random Forest, and *mtry*, the number of features randomly selected for consideration at each split, are held constant across all methods and both stages at 500 and 4, respectively. The shared parameter values for the first stage are as follows:

- *nodesize*: the minimum size of a terminal node within a Decision Tree
    - *From*: 10; *To*: 50; *By*: 10
- *cutoff*: the minimum proportion (inclusive) of votes for the minority class required for a positive final prediction
    - *From*: 0.1; *To*: 0.5; *By*: 0.1
- *over:* the amount of oversampling performed, expressed as a percentage of the original size of the minority class training set, ie. 100% oversampling results in a minority class training set 200% its original size
    - *From:* 0%; *To*: 500%; *By*: 100%
- *under:* the amount of undersampling performed, expressed as a percentage of the original size of the majority class training set, ie. 100% undersampling results in the complete removal of the majority class examples
    - *From*: 0%; *To* 80%; *By*: 20%
- *k:* the number of neighbors searched for in nearest-neighbor calculations
    - *From*: 1; *To*: 5; *By*: 2
- *cost* (for **CSL** only): the cost of misclassifying positive examples (cost of misclassifying negative examples is held constant at one)
    - *From*: 1; *To*: 10; *By*: 1

6.2 Results of Optimization: Computational Expense

The results of the preliminary stage of optimization are shown in Table 4. While we are ultimately concerned with the performance of our models in terms of the Gmean, in this stage we are also concerned with computational expense, expressed as the "Mean Time." The mean time does not refer to the average time to compute the cross-validation function for all attempted parameter values; rather, it is the average computational time of five executions of

Table 4: First round of optimization

Method	Gmean	TNR	TPR	*nodesize*	*cutoff*	*over*	*under*	*k*	*cost*	Mean Time
RF	62.30	54.58	71.12	10	0.1	-	-	-	-	0.72s
RUSROS	65.44	55.10	77.71	30	0.2	200%	0%	-	-	0.78s
OSS	62.94	60.59	65.39	40	0.4	200%	-	-	-	14.85m
SMOTE	67.51	64.09	71.11	20	0.2	100%	0%	5	-	1.17s
STMK	64.60	69.28	60.24	20	0.3	100%	-	1	-	4.87m
CSL	65.92	61.37	70.81	50	0.4	-	-	-	6	0.55s
SBAG	62.11	66.41	58.10	30	0.3	100%	20%	5	-	2.13m
SBST	62.77	61.95	63.60	30	0.3	200%	0%	3	-	2.50m
HDRF	42.73	19.92	91.67	10	0.1	-	-	-	-	14.71s
BSI	65.73	72.00	60.00	-	-	-	-	-	-	-

Table 5: Second round of optimization

Method	Gmean	TNR	TPR	*nodesize*	*cutoff*	*over*	*under*	*k*	*cost*
RF	65.27	54.82	77.71	6	0.11	-	-	-	-
RUSROS	67.44	67.26	67.62	5	0.15	50%	0%	-	-
SMOTE	67.87	64.56	71.35	25	0.2	100%	0%	7	-
CSL	66.46	62.38	70.81	46	0.4	-	-	-	6
HDRF	67.02	64.61	69.52	1	0.14	-	-	-	-
BSI	65.73	72.00	60.00	-	-	-	-	-	-

the cross-validation function, using the optimal parameter values as inputs. In this way, this statistic does not represent a thorough analysis of the computational expense of each method, but does provide for a quick comparison of scale. In making comparisons, it is important to be aware of the effects different parameter values can have on the computational time. For example, considering the same function was used for the base Random Forest model and Cost-Sensitive Learning, it is unlikely that **CSL** is actually faster than **RF**, and only appears to be so because the minimum node size for the optimal **CSL** model was much larger, thus its trees were grown shorter and therefore quicker. Additionally, larger oversampling percentages slow down the execution of the SMOTE and One-Sided Selection methods considerably. Noting the weaknesses of this measure, it is primarily given as a way to identify methods which are particularly slow to compute, namely: One-Sided Selection; SMOTE-Tomek; SMOTEBagging; and SMOTEBoost. Of these, it should be unsurprising that the ensemble methods are slow: instead of performing SMOTE ten times per cross-validated estimate, SMOTEBagging and SMOTEBoost must perform SMOTE 5,000 and 1,000 times, respectively. One-Sided Selection and SMOTE-Tomek, on the other hand, seem unreasonably slow; the problem here likely lies not only with the computational expense of these methods but also with inefficiencies in the code for the One-Sided Selection function. In any case, all four of these methods were not optimized beyond the first stage as a result of their slow computational times.

For the second stage of optimization, the results of which are given in Table 5, optimization was still performed globally, but step size was tailored to each function depending on how many parameters needed to be optimized for the respective method. Consider the example of SMOTE: with a minimum step size of 1/500 trees

and an upper bound of 1/2 for the cutoff, there are 250 possible values for the vector of cutoffs; assuming an average of 90% of the minority training examples present in any training set for ten-fold cross validation, there are $70 \times 0.9 = 63$ possible oversampling percentages up to 100%, plus an additional four if the Imbalance Ratio of 5.71 is taken as an upper bound; similarly, $400 \times 0.9 = 360$ possible values exist for the undersampling percentage; ignoring the vectors of values for the node size and k, for which determining upper bounds is more of a qualitative decision, optimizing these three vectors alone, globally, and with their minimum step size, results in a search grid of 6.03×10^6 combinations. Such tasks are best left to supercomputers. The modeler posed with this problem can take one or both of two compromises: either they can maintain a low step size and search locally among values close to the previously optimal ones, or they can search globally but decrease the step size to a lesser extent. We have chosen the second of these options. Describing in detail the discretionary decisions made for the parameter searches of each method in the second stage of optimization would be tedious, but to give the reader an idea: for SMOTE, the step sizes of the cutoff, minimum node size, and undersampling percentage were reduced by one half, while the range of k was increased to nine and the range of oversampling percentages was expanded to include values under 100%, resulting in a search grid of 45,000 combinations. This represented the largest search grid, while **RF** and **HDRF** had the smallest at 2,500 combinations each, resulting from 50 values tried for the node size and cutoff.

6.3 Results of Optimization: Accuracy

The first round of optimization already demonstrates the power of Random Forests in handling imbalanced data. Not only do the **SMOTE** and **CSL** models already outperform the original **BSI** model, but the base **RF** model itself does surprisingly well with appropriate choices for the minimum node size and cutoff. **HDRF**, on the other hand, at first appears to be a very poor model, with a Gmean below 50%.

The ensemble methods **SBAG** and **SBST** have yielded lackluster results, with the Gmean of **SBAG** falling below that of the base **RF** model and **SBST** scoring slightly above it. Without further optimization, it is impossible to draw conclusions about how these models would rank against the others, although their accuracies would no doubt improve. We can, however, speculate as to why they have performed poorly in the first stage of optimization. For SMOTEBagging, for example, the implementation of random feature selection before constructing the trees is likely not as robust as randomly selecting a subset of features at each split. On the other hand, for SMOTEBoost, two possible reasons may have reduced accuracy: firstly, using Random Forests as the weak learners was computationally expensive, and in the interest of time ensembles of only 100 Random Forests were grown, which may have been too small. Secondly, Boosting works by shifting the decision boundary towards examples which are difficult to classify. Beyond the general risk of any data set to possess noisy examples, the relatively small size of our minority class training set may increase this risk, as there may be isolated examples which could prove to be members of small clusters if the sample size were larger. Additionally, the SMOTE algorithm itself runs the risk of generating noisy examples if there are already noisy examples present in the data or if the value of k is chosen to be too large, as new data can be synthesized between clusters, between noisy examples, or between a noisy example and a cluster. The Boosting algorithm, instead of ignoring the noise, will focus more and more on these impossible to classify examples. Neither **SBAG** or **SBST**

performs altogether poorly, however, and if the above-mentioned issues do, in fact, detract from their accuracies, the extent to which they do so is likely not great.

Other interesting observations can be drawn by comparing the different data-level preprocessing methods. It seems that, for this data set, oversampling proves to be a much stronger approach than undersampling, for three reasons: firstly, because **SMOTE** was the best performing method of all; secondly, because the optimal models for the resampling methods for which undersampling was optional chose, with the exception of **SBAG**, to perform no undersampling at all; and thirdly, because **STMK** outperformed **OSS**, but **RUSROS** with 0% undersampling outperformed both of these. It is also interesting to note that, where oversampling was performed, the oversampling percentages for the optimal models lie well below the Imbalance Ratio, leaving their training sets still quite imbalanced in favor of the majority class.

The second round of optimization improved the results of all retained methods, such that all but the base **RF** model outperformed the original **BSI** model. Here, **RUSROS** and **SMOTE** were the best performing methods, again with no undersampling performed and with only a small amount of oversampling. This seems to indicate that data-level preprocessing approaches are good solutions when faced with imbalanced data, but that they should not be used to completely eliminate imbalance in the data set. The method which benefitted the most from the additional optimization was **HDRF**, whose Gmean increased by 24.29% to beat out **CSL** for the third-best method. Here, the optimal model contained only pure end nodes, indicating a considerable improvement to the splitting function. The example of **HDRF** demonstrates the importance of proper optimization when judging the quality of a model. With the exception of **RUSROS**, whose optimal minimum node size in the second stage of optimization differed significantly from that of the first, most of the optimal models could have been found with a localized parameter search centered on the optimal values from the first stage. It is likely that most of the methods would have seen greater increases in their Gmean estimates had the second stage of optimization been performed locally and with a smaller step size.

7. Conclusion

In this paper, we have demonstrated the strength of Random Forests in handling imbalanced data, owing to the control they give the modeler over generalization (minimum node size) and bias towards the minority class (vote threshold). Of the variety of methods applied to the problem, SMOTE performed best, and oversampling methods appeared to perform particularly well, even though the optimal amount of oversampling often preserved a high level of imbalance in the training set. With proper optimization, all methods, aside from the base Random Forest model, outperformed the original Boosted-SVM for Imbalanced Data method. Further work could involve additional localized optimization with reduced step sizes for all methods, including the ensemble- and undersampling-based methods left out of the second stage of optimization. Additionally, subsets of the data could be created by incrementally raising the minimum information gain requirement from Table 1 in order to explore the effects of dimensionality on the accuracy of the models. Lastly, **HDRF** could be applied to additional datasets to test whether trees of maximum size should always be grown, which would reduce the number of parameters to optimize for this method to one.

References

1. Abdi, L., & Hashemi, S. (2016). To combat multi-class imbalanced problems by means of over-sampling techniques. *IEEE Transactions on Knowledge and Data Engineering*, 238-251.

2. Barandela, R., Valdovinos, R., & Sánchez, J. (2003). New applications of ensembles of classifiers. *Pattern Analysis*, 245-256.

3. Barua, S., Islam, M., Yao, X., & Murase, K. (2014). MWMOTE-majority weighted minority oversampling technique for imbalanced data set learning. *IEEE Transactions on Knowledge and Data Engineering*, 405-425.

4. Batista, G., Bazzan, A., & Monard, M. (2003). Balancing Training Data for Automated Annotation of Keywords: a Case Study. *Revista Tecnologia da Informação*, 15-20.

5. Batista, G., Prati, R., & Monard, M. (2004). A study of the behaviour of several methods for balancing machine learning training data. *SIGKDD Explorations*, 20-29.

6. Bhattacharyya, A. (1943). On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*, 99-109.

7. Błaszczynski, J., Deckert, M., Stefanowski, J., & Wilk, S. (2010). Integrating selective pre-processing of imbalanced data with ivotes ensemble. *Rough Sets and Current Trends in Computing* (pp. 148-157). Berlin/Heidelberg: Lecture Notes in Computer Science.

8. Boonchuay, K., Sinapiromsaran, K., & Lursinsap, C. (2017). Decision tree induction based on minority entropy for the class imbalance problem. *Pattern Analysis*, 769-782.

9. Branco, P., Ribeiro, R. P., & Torgo, L. (2016). UBL: an R Package for Utility-Based Learning. Retrieved from http://arxiv.org/abs/1604.08079

10. Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 123-140.

11. Breiman, L. (1996). *Out-of-bag estimation.* Retrieved from https://www.stat.berkeley.edu/~breiman/OOBestimation.pdf

12. Breiman, L. (2001). Random Forests. *Machine Learning*, 5-32.

13. Breiman, L., Friedman, J., Stone, C., & Olshen, R. (1984). *Classification and Regression Trees.* Taylor & Francis.

14. Bunkhumpornpat, C., Sinapiromsaran, K., & Lursinsap, C. (2009). Safe–level–SMOTE: safe–level–synthetic minority over–sampling TEchnique for handling the class imbalanced problem. *Proceedings of the 13th Pacific–Asia Conference on Advances in Knowledge Discovery and Data Mining*, (pp. 475-482). Bangkok.

15. Bunkhumpornpat, C., Sinapiromsaran, K., & Lursinsap, C. (2012). DBSMOTE: density-based synthetic minority over-sampling TEchnique. *Applied Intelligence*, 664-684.

16. Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, W. (2002). SMOTE: synthetic minority oversampling technique. *Journal of Artificial Intelligence*, 321-357.

17. Chawla, N., Lazarevic, A., Hall, L., & Bowyer, K. (2003). SMOTEBoost: improving the prediction of the minority class in boosting. *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases* (pp. 107-119). Berlin/Heidelberg: Springer.

18. Chen, C., Liaw, A., & Breiman, L. (2004). Using Random Forest to Learn Imbalanced Data. University of California, Berkeley.

19. Cieslak, D., & Chawla, N. (2008). Learning decision trees for unbalanced data. *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, (pp. 241-256). Antwerp.

20. Cieslak, D., Hoens, T., Chawla, N., & Kegelmeyer, W. (2012). Hellinger distance decision trees are robust and skew-insensitive. *Data Mining and Knowledge Discovery*, 136-158.

21. Cohen, G., Hilario, M., Sax, H., Hugonnet, S., & Geissbuhler, A. (2006). Learning from imbalanced data in surveillance of nosocomial infection. *Artificial Intelligence*, 7-18.

22. Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification . *IEEE Transactions on Information Theory*, 21-27.

23. Dua, D., & Karra Taniskidou, E. (2018). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. Retrieved from http://archive.ics.uci.edu/ml

24. Fawcett, T. (2006). An Introduction to ROC Analysis. *Pattern Recognition*, 861-874.

25. Fernandéz, A., García, S., Galar, M., Prati, R., Krawczyk, B., & Herrera, F. (2018). *Learning from Imbalanced Data Sets.* Cham: Springer.

26. Freund, Y., & Schapire, R. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and Systems Sciences*, 119-139.

27. Galar, M., Fernández, A., Barrenechea, E., & Herrera, F. (2013). Eusboost: enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. *Pattern Recognition*, 3460-3471.

28. Han, H., & Wang, W. M. (2005). Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. *Proceedings of the 2005 International Conference on Intelligent Computing* (pp. 878-887). Hefei: Lecture Notes in Computer Science.

29. Hart, P. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 515-516.

30. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning.* New York: Springer.

31. He, H., Bai, Y., & Garcia, E. L. (2008). ADASYN: adaptive synthetic sampling approach for imbalanced learning. *Proceedings of the 2008 International Joint Conference on Neural Networks*, (pp. 1322-1328). Hong Kong.

32. Hellinger, E. (1909). Neue Begründung der Theorie quadratischer Formen von unendlichvielen Veränderlichen. *Journal für die reine und angewandte Mathematik*, 210-271.

33. Ho, T. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 832-844.

34. Hu, S., Liang, Y., Ma, L., & He, Y. (2009). MSMOTE: Improving classification performance when training data is imbalanced. *2nd International Workshop on Computer Science and Engineering*, (pp. 13-17). Qingdao.

35. Krawczyk, B. (2016). Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 221-232.

36. Kubat, M., & Matwin, S. (1997). Addressing the Curse of Imbalanced Training Sets: One-Sided Selection. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 176-189). San Francisco: Morgan Kaufmann.

37. Kuhn, M. (2018). caret: Classification and Regression Training. Retrieved from https://CRAN.R-project.org/package=caret

38. Laurikkala, J. (2001). Improving identification of difficult small classes by balancing class distribution. *Proceedings of the 8th Conference on AI in Medicine in Europe*, (pp. 63-66). Cascais.

39. Lenca, P., Lallich, S., Do, T., & Pham, N. (2008). A comparison of different off-centered entropies to deal with class imbalance for decision trees. *Proceedings of the 12th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, (pp. 634-643). Osaka.

40. Liaw, A., & Wiener, M. (2002). Classification and Regression by randomForest. Retrieved from https://CRAN.R-project.org/doc/Rnews/

41. Liu, W., Chawla, S., Cieslak, D., & Chawla, N. (2010). A robust decision tree algorithm for imbalanced data sets. *Proceedings of the SIAM International Conference on Data Mining*, (pp. 766-777). Columbus.

42. Liu, X., Wu, J., & Zhou, Z. (2009). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 539-550.

43. Menardi, G., & Torelli, N. (2014). Training and assessing classification rules with imbalanced data. *Data Mining and Knowledge Discovery*, 92-122.

44. Mishina, Y., Tsuchiya, M., & Fujiyoshi, H. (2014). Boosted Random Forest. *Proceedings of the International Conference on Computer Vision Theory and Applications*, (pp. 594-598). Lisbon.

45. Orriols-Puig, A., & Bernadó-Mansilla, E. (2009). Evolutionary rule-based systems for imbalanced data sets. *Soft Computing*, 213-225.

46. Quinlan, J. (1986). Induction of Decision Trees. *Machine Learning*, 81-106.

47. Quinlan, J. (1993). *C4.5: Programs for Machine Learning.* San Francisco: Morgan Kaufmann.

48. Ramentol, E., Caballero, Y., Bello, R., & Herrera, F. (2012). SMOTE-RSB*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using smote and rough sets theory. *Knowledge and Information Systems*, 245-265.

49. Robnik-Sikonja, M., & Savicky, P. (2018). COREIearn: Classification, Regression, and Feature Evaluation. Retrieved from https://CRAN.R-project.org/package=COREIearn

50. Sáez, J., Luengo, J., Stefanowski, J., & Herrera, F. (2015). SMOTE-IPF: addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering. *Information Science*, 184-203.

51. Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PloS One*.

52. Schaffer, C. (1993). Selecting a Classification Method by Cross-Validation. *Machine Learning*, 135-143.

53. Schapire, R. (1990). The strength of weak learnability. *Machine Learning*, 197-227.

54. Seiffert, C., Khoshgoftaar, T., Van Hulse, J., & Napolitano, A. (2010). Rusboost: a hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man, and Cybernetics*, 185-197.

55. Stefanowski, J., & Wilk, S. (2008). Selective pre-processing of imbalanced data for improving classification performance. *Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery*, (pp. 283-292). Turin.

56. Tang, S., & Chen, S. (2008). The generation mechanism of synthetic minority class examples. *Proceedings of the Fifth International Conference on Information Technology and Applications in Biomedicine*, (pp. 444-447). Shenzhen.

57. Therneau, T., & Atkinson, B. (2018). rpart: Recursive Partitioning and Regression Trees. Retrieved from https://CRAN.R-project.org/package=rpart

58. Ting, K. (2002). An instance-weighting method to induce cost-sensitive trees. *IEEE Transactions on Knowledge and Data Engineering*, 659-665.

59. Tomek, I. (1976). Two Modifications of CNN. *IEEE Transactions on Systems, Man and Communications*(SMC-6), 769-772.

60. Wang, S., & Yao, X. (2009). Diversity analysis on imbalanced datasets by using ensemble models. *IEEE Symposium on Computational Intelligence and Data Mining*, (pp. 324-331). Nashville.

61. Wilson, D., & Martinez, T. (1997). Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research*, 1-34.

62. Wright, M. N., & Ziegler, A. (2017). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Journal of Statistical Software*, 1-17. doi:10.18637/jss.v077.i01

63. Yang, Y., & Pederson, J. (1997). A comparative study on feature selection in text categorization. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 412-420). Morgan Kaufmann.

64. Yen, S., & Lee, Y. (2006). Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset. *Proceedings of the International Conference on Intelligent Computing* (pp. 731-740). Kunming: Lecture Notes in Control and Information Sciences.

65. Yen, S., & Lee, Y. (2009). Cluster-based under-sampling approaches for imbalanced data distributions. *Expert Systems with Applications*, 5718-5727.

66. Yoon, K., & Kwek, S. (2005). An unsupervised learning approach to resolving the data imbalanced issue in supervised learning problems in functional genomics. *Proceedings of the Fifth International Conference on Hybrid Intelligence Systems*, (pp. 303-308). Rio de Janeiro.

67. Zhang, J., & Mani, I. (2003). KNN approach to unbalanced data distributions: a case study involving information extraction. *Proceedings of the 20th International Conference on Machine Learning.*

68. Zięba, M., Tomczak, J. M., Lubicz, M., & Świątek, J. (2014). Boosted SVM for extracting rules from imbalanced data inapplication to prediction of the post-operative life expectancy in the lung cancer patients. *Applied Soft Computing*, 99-108.