ConfDTree: Improving Decision Trees Using Confidence Intervals

Gilad Katz, Asaf Shabtai, Lior Rokach, Nir Ofek
Department of Information Systems Engineering and Telekom Innovation Labs,
Ben-Gurion University of the Negev
Beer-Sheva, Israel
{katzgila, shabtaia, liorrk, nirofek}bgu.ac.il

Decision trees have three main disadvantages: reduced performance when the training set is small, rigid decision criteria and the fact that a single "uncharacteristic" attribute might "derail" the classification process. In this paper we present ConfDTree - a post-processing method which enables decision trees to better classify outlier instances. This method, which can be applied on any decision trees algorithm, uses *confidence intervals* in order to identify these hard-to-classify instances and proposes alternative routes. The experimental study indicates that the proposed post-processing method consistently and significantly improves the predictive performance of decision trees, particularly for small, imbalanced or multi-class datasets in which an average improvement of 5%-9% in the AUC performance is reported.

Keywords-decision trees, confidence intervals, imbalanced datasets

I. INTRODUCTION

The area of decision trees is probably one of the most extensively researched domains in machine learning. Aside from advantages such as the ability to explain the decision process and low computational costs, decision trees also usually produce relatively good results in comparison to other machine learning algorithms. Although the most popular decision trees induction algorithms, such as C4.5 and CART, were developed a long time ago, they are still frequently used for solving everyday classification tasks. In this paper we aim to improve the predictive performance of these algorithms by mitigating three of their main drawbacks:

(a) *Reduced performance when the training set is small* – Small sample sizes pose a great challenge to decision trees [1], in particular because the number of available training instances drops exponentially as the tree branches out (and the number of leaves is bounded by the training set size). For this reason, if the training set is too small, the induction algorithm may grow an overly simplistic classification tree.

(b) *"Rigid" decision criteria* – the decision at each level (node) of the tree is rigid in the sense that only one branch (node) can be chosen (unless the classified instance has no value in the attribute, which is a different problem). This approach usually works well, but consider the following scenario: the performed test on attribute X is X≤10, and the value of attribute X of a given instance is 9.99. In such cases, should we not at least consider the possibility that the classification might be incorrect, or consider an alternative path?

This problem is not new and has been discussed almost 20 years ago in [2] ("soft thresholds"). In other classification algorithms the problem has been addressed by using the notion of margin. One particular case is Support Vector Machines, where instead of using a fixed hyperplane for separating classes, the algorithm uses varying margins in order to better classify the dataset.

(c) *Outlier attribute values* – deep decision trees consisting of many levels might also include many attributes. Such bushy trees are often seen in particular when the training set is large in terms of number of instances and number of attributes. The need to rely on so many attributes to reach a classification leads to the following potential problem: it only takes one attribute with an outlier value (one that is uncharacteristic of its class) to "derail" the classification process.

This claim can be easily supported using the following scenario. We are given a tree with a depth of h levels. Let $\alpha_i(x)$ be the probability of selecting the wrong branch for a given instance x on node l. For the sake of simplicity, let assume that $\alpha_i(x) \geq \varepsilon$ where $\varepsilon > 0$, and that the errors are independent. Thus, the probability for reaching the wrong leaf is $\Pr(err) = 1 - (1 - \varepsilon)^h$ which of course increases with h. Moreover, in many cases $\alpha_i(x)$ increases as node l is located in a deeper level, mainly because the number of available training instances drops exponentially as the tree branches out.

These problems are considerably aggravated in imbalanced datasets [3], where there are many more instances of a certain class than of another. In such cases, standard machine learning techniques may be "overwhelmed" by the majority class and ignore the minority class. Combined with the lack of sufficient number of instances, outlier values make it even more difficult to correctly classify under-represented classes. In order to enable decision trees to address imbalanced datasets two practices are frequently used:

1. Instead of using the leaf classification just as in balanced classification tasks, in imbalanced tasks it is better to use the classification distribution associated with the designated leaf. This can be used to rank the test instances according to their probabilities and help the decision maker to select the best trade-off between the true-positive and false-positive performance. As a result, reaching to the incorrect leaf might have a greater impact. In balanced classification tasks designating the wrong leaf might still result in the same classification

output (when the derailing is targeted to the incorrect leaf but with the same classification). However in imbalanced tasks, derailing to the incorrect leaf will usually result in a different classification distribution and therefore affect the test instances ranking.

2. Due to the fact that the classification distribution is frequently used in imbalanced tasks, it is recommended to avoid pruning [4]. Hence, classification trees for imbalanced tasks usually have more levels and are therefore prone to over-fitting.

Similar to imbalanced datasets, multi-class problems also pose a challenge for classification trees, particularly when the number of classes goes beyond a 'modest' size [5]. This can be explained by the fact that in multi-class tasks, derailing a test instance from its correct leaf will results in a wrong classification. This is not true for balanced binary classification tasks, in which derailing to an incorrect and random leaf may still result in a correct classification with a probability equal to the percentage of the item's class of the total "population".

These problems may be avoided by the use of ensemble methods like RandomForest [6]. By creating a large number of trees with varying attributes, outlier attributes and borderline decisions become less of an obstacle. However, these methods require both long training and execution times and large amounts of memory. In addition, decision forests are considered to be less comprehensible than a single decision tree.

In this paper we propose a post processing method to address these three problems; namely, decision trees with confidence intervals. We use simple statistical measures in order to determine whether the decision made at every level of the tree has a minimum degree of plausibility with respect to the final outcome (i.e., classification of an instance in a leaf). If this is not the case, we employ one of two solutions; alternative routes or certainty fines, in order to explore other classification possibilities.

The core idea of this paper is as follows. Once a classification has been made, we follow the decision path from the leaf back to the root, examining every decision made along the way. After the examined instance has been assigned to a class, our goal is to determine whether the attributes that were used to classify the instance can be said to be "characteristic" of that class. If that is not the case, the solutions mentioned above, alternative paths and certainty fines are used in order to consider other possible classifications.

In order to determine whether an attribute with a certain value "belongs" to a certain class we use confidence intervals. Confidence intervals enable us to determine with a predefined level of certainty whether or not the value of the examined attribute is within a range that can be considered as belonging to the class in question. The confidence intervals are easy to compute and do not add additional complexity to the overall computational effort of the classification process.

One of the advantages of the proposed method is the fact that it does not interfere with the generation of the decision tree. This means that although our experiments were conducted using trees generated by the C4.5 algorithm, the proposed method can be applied on decision trees produced by other algorithms.

The rest of this paper is organized as follows. Section 2 introduces related work of variations of decision trees and decision trees that are combined with statistical methods. In Section 3 we present the proposed ConfDTree and in Section 4 we evaluate it. Lastly, Section 5 concludes the paper.

II. Related Work

A. Decision Trees

Decision trees are directed graphs used to classify items. They consist of a *root node* (a node in the graph to which no other node points), *internal nodes* (nodes that are pointed at and point to other nodes) and *leaves* (nodes that do not point to other nodes). During the classification process, the classified item "travels" from the root to one of the leaves, where a classification is made. The classification may simply be one of the possible classes or a set of probabilities (one for each of the possible class values).

At this point, we wish to provide definitions for several base terms: (a) *immediate descendant* node – if a node points to other nodes in the tree, these nodes are its "immediate descendants"; (b) *split attribute* – the attribute by which a decision is made at some node in the tree; (c) *split value* – the value of the split attribute which determines to which immediate descendant node the classified instance will be assigned (for example, if $X \leq 10$ is the condition for one immediate descendant node and $X > 10$ is the condition for another, then 10 is the split value; (d) *class* – the class is the attribute whose values are what we attempt to predict during the classification phase.

The algorithms for the generation of decision trees are numerous and many methods are used for this purpose. All algorithms use recursive partitioning, but they usually differ in the manner of how they choose the attribute by which to split each node in the tree, as well as by the stopping criteria (the decision not to perform additional splits on a node). The ID3 [7] and C4.5 [8] algorithms, for example, use the Information Gain and Gain Ratio measures respectively and also differ on the tests performed on their attributes. The CART algorithm [9] uses the Gini Impurity measure [10] and regression in the leaves in order to produce its prediction.

Contrary to the methods described above which build a single decision tree other methods use many. Algorithms such as RandomForest [6] and RotationForest [11] create many subsets of training instances and attributes and use them to train multiple trees. During the classification phase, each tree provides its prediction and they are all combined into one. These methods usually produce superior results when compared to those that use single trees (partly because they provide a solution, at least to some degree, to the drawbacks mentioned in Section 1), but they require large amounts of computing resources and are not comprehensible to users. In addition, the dataset requires a sufficiently large number of attributes.

B. Combinations of decision trees and statistics

The problem of outliers is not new in the field of decision trees, especially in unbalanced datasets. However, previous work was focused on the tree generation phase. Most works were aimed at preventing the outliers from influencing the creation of the model and on improving probabilities estimation. For example, John [12] proposed iterative removal of instances with outlier values, while Last *et al*. [13] used statistical methods in order to create a more robust classifier. Zadorny and Elkan [14], and Provost and Domingos [4] and [15] focused on improving the classification probabilities produced by the tree by the use of regression in the leaves. The use of confidence measures is also not uncommon; McCallum [16], for instance, used the *Kolmogorov-Smirnov test* [17] in order to determine whether additional nodes in tree should be created based on the differences in the distribution of instances from different class. These researches, however, propose methods for ignoring outliers during the creation of the model, but offer no solutions for correcting classifications during the test phase.

Another well-known method for dealing with outliers is fuzzy decision trees [18, 19]. By using fuzzy functions to construct the tree and by varying the degrees of certainty based on the attributes of the classified instance, these methods offer a greater degree of flexibility in dealing with outliers whose values are slightly irregular (achieved by using multiple states and weights). This fuzziness offers a possible solution to the "rigidness" problem mentioned above; however, there are several substantial differences between this method and the one proposed in this paper: (1) the fuzzy function may be domain specific and require a human expert in order to correctly define it. Our method, on the other hand, uses simple statistical tools that require very little (if any) tuning; (2) fuzzy decision trees will have difficulty in correctly classifying instances whose outlier attribute values differ greatly from the norm (as they will be out of the scope of the fuzzy function); and (3) the proposed method differs from fuzzy decision trees because it modifies the original classification *only* when it is very likely that the classification is incorrect; i.e., knowing that the attribute value is borderline is not a sufficient cause to act and therefore, we also verify that the value is out of the norm of the assigned class.

The proposed method contains several similarities to the work presented above. We use statistical measures, and specifically confidence intervals, with the goal of the proposed algorithm being to deal with outliers. However, unlike any of the methods presented above, our method is a post processing method applied *after* the decision tree has already been derived, during the classification phase. This difference means that the plausibility of the classification is verified at various points along the classification path, increasing the probability of correctly handling outliers. In addition, although "smoothing" has been used and applied on the leaves of a decision tree [20], it has not been used in the inner nodes or in any other way than "tweaking the results". Lastly, to the best of our knowledge confidence intervals have not previously been used for improving classifiers and decision trees in particular. Note that our goal is not to assign a confidence score to the classifier's predictions, a concept that has been investigated before [21], but rather to improve the classification itself.

III. The Proposed Method

In this section we present the proposed method. We explain the required calculations needed during the generation of the tree and go over the proposed modifications to the classification process.

A. Deriving the Confidence Intervals

Confidence intervals are used to assess the reliability of an estimate. They enable us to define a range of values within which an instance, randomly sampled from a distribution, is likely to be. In other words, a confidence interval with a probability of $(1-\alpha)$ means that a random instance from the distribution has a $(1-\alpha)$ chance of being inside it, whereas α is the probability of the instance being outside the interval. For example, in normal distributions the confidence interval can be defined by $[\overline{X}-z\cdot\sigma\ ,\ \overline{X}+z\cdot\sigma]$ where $\overline{X}$ is the average of the distribution, z is the corresponding value in the Z table for the requested significance level of the interval and σ is the standard deviation.

According to the proposed method, once the decision tree has been generated, the following values are calculated for every internal node:

1. the average value of the split attribute
2. confidence intervals for values of the split attribute

This is done separately for every class whose training set instances are included in the node.

There are two prerequisites for the generation of the confidence intervals:

1. The split attribute of the node should be numeric;
2. A minimal number of instances m (a predefined threshold) exists for every class present in the node.

We experimented with three different approaches for calculating the confidence intervals: one based on the t distribution, one based on the normal (Z) distribution and the third with both distributions combined (their application is explained later on).

The t distribution is suitable for our needs because it enables us to make an inference regarding the mean when the standard deviation is unknown. In addition, it assumes that the distribution values are more dispersed than in a normal distribution, a fact which enables it to be suitable for a larger range of scenarios. The normal distribution, however, is meant to be used on values that are normally distributed. For data that is known to be normally distributed, calculating the confidence interval using the normal (Z) distribution will be more accurate.

However, for obvious reasons, we cannot assume that the split attribute values are always normally distributed and therefore we used the Kolmogorov-Smirnov (K-S) test [17] on every node with a numeric split attribute. The following calculations (and subsequently, the algorithm presented here) were only applied on nodes for which all present class

attributes "passed the test" (the K-S test was applied separately for each class attribute).

We hypothesized that the normal distribution (when applicable) would be more accurate than the *t* distribution, but we were concerned that there could be datasets whose attributes would not be distributed in such a way. We therefore decided to test an approach that uses the normal distribution to generate the confidence intervals whenever possible and uses the *t* distribution in all other cases. We call this approach the combined approach.

A general example of calculating these measurements is presented in Fig. 1. In this example the split attribute in the highlighted node is *Y* and the possible classes are *A, B* and *C*, where 100 instances from the training set are of class *A*, 100 are of class *B*, and 30 are of class *C*. The average of the values of attribute *Y* of instances that belong to class *A* is 4.75 and the confidence interval is [3,6.5].

The pseudo code for deriving the confidence intervals is presented in Alg. 1. For each node the algorithm first checks if it is a leaf or not (line 1). If the node is a leaf, or if the number of instances in the node is smaller than a predefined threshold m (or if the values of the node are not normally distributed when only the *Z* distribution is used), the procedure terminates; otherwise, the average and standard deviation of the split attribute are calculated separately for each class (lines 2-5). Then, the procedure is recursively run for each of the immediate descendant nodes (lines 6-7).

The additional computations during the training phase include the calculation of the average and standard deviation of the split attribute for each class in every inner node. The computational complexity is therefore $O(n \cdot (d\text{-}1))$, where n is the number of instances in the tree, and d is the number of levels in the tree (height of the tree).

B. The Classification Phase

During the classification phase, we first use the induced tree to provide a classification distribution. That is, the examined instance traverses the tree top-down to one of its leaves, thus producing a classification. This classification consists of the probabilities of the classified instance belonging to each of the class values, whose sum is always 1. For example, in Fig. 2 it can be seen that the probabilities produced by the decision tree (leaf number 1) are {1,0,0} for class *A*, *B* and *C* respectively, which indicates that the tree is 100% certain of its classification. Using the class frequency in the tree leaves as-is will typically overestimate the probability. In order to avoid this phenomenon, it is useful to perform the Laplace correction [22]. Later, in Section 3.2.2 we suggest a variation of the Laplace correction which better fits our goals.

At this point we would like to define an additional term – *assigned class*. This term refers to the class which has the highest probability. For example, if the probabilities of classes *A*, *B* and *C* are 0, 0, and 1 respectively, then *C* will be defined as the assigned class. We will use this term from this point on.

After the classification probabilities have been obtained, the classified instance “travels” up the tree, back to the root. For each of the nodes in its path, not including the leaf, we apply the following algorithm:

1) If the instance is within the confidence interval of the assigned class - *do nothing*. The instance is said to be *within the confidence interval of the assigned class* if the instance’s value of the node’s split attribute is within the confidence interval of the split attribute of the assigned class. We consider this to be the “normal” situation where the value of the split attribute of the instance is within the area considered as “likely”.
2) If the instance is not within the confidence interval of the class it is currently assigned to, there are two options:
   a) It is within the confidence interval of another class (or classes) – in this case we generate *alternative routes*.
   b) It is not within the confidence interval of any other class – in this case we impose certainty fines.

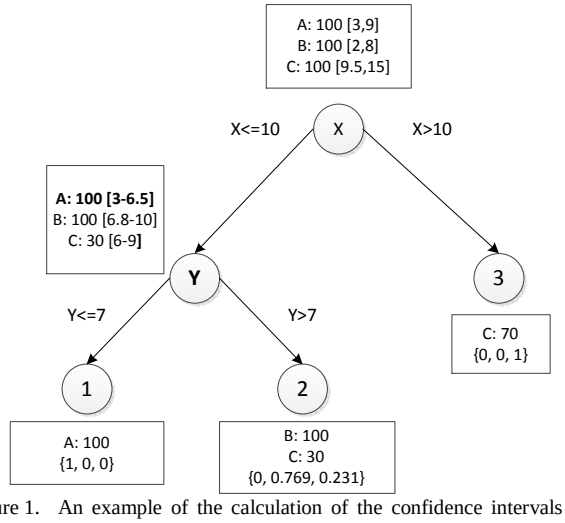

Figure 1. An example of the calculation of the confidence intervals for each class in the internal nodes. Squared brackets in the internal nodes indicate the confidence intervals. The values in the curly brackets indicate the classification distribution vector.

Algorithm 1 Deriving the confidence intervals

Input: Node node: a node in the tree
List<Instance> instances: all the training instances in the node

```
1: IF Is_Leaf(node)==true or node.size < m THEN RETURN
2: split_attribute ← node.split_attribute;
3:node.num_of_instances ← Get_Num_Of_Instances_Per_ClassID(node);
4. node.attribute_averages ← Calculate_Attribute_Average_By_ClassID;
5. node.stdevs ← Calculate_Stdev_Per_Class_ID(instances,
                    num_of_instances, attribute_averages)
6. Node[] sons ← Get_Son_Nodes(split_attribute)
7. FOR (int i=0; i<sons.Length; i++) DO
        Build_Tree_With_Confidence_Intervals(sons[i],
        Find_Instances_That_Match_Criteria(instances, split_criteria)
8: END FOR
```

These courses of action can best be explained by the example shown in Fig. 3. After being classified to leaf 1, the classified instance "travels" bottom-up the tree (note the direction of the arrows). The number of instance from each class and their confidence intervals are presented next to

their corresponding internal nodes (the relevant confidence intervals in bold). In the internal node *Y*, the confidence interval of class *A* (the current assigned class) is the only one that contains the instance's attribute value. In the root node, however, the instance's attribute value is outside the confidence interval of *A* (it is, however, inside the confidence interval of class *C*).

In the following sub-sections we will go over these two scenarios in detail.

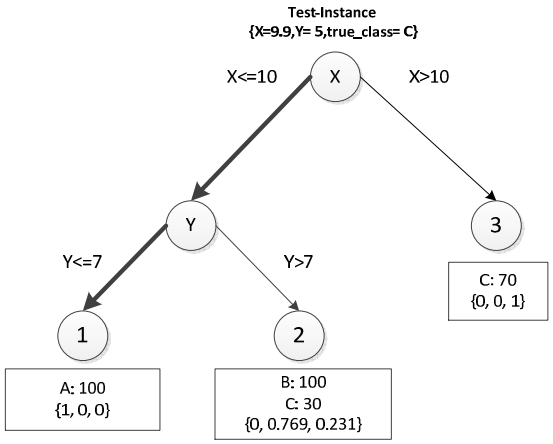

Figure 2. An example for the classification process of a C4.5 decision tree. The attribute values of the classified instance, whose true class is C, are presented above the root node and correspond with X, Y and the true class. The probabilities of each class in the leaves is denoted by the curly brackets.

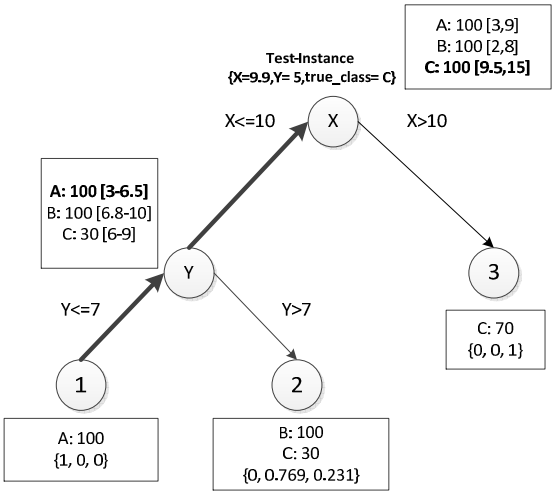

Figure 3. After its classification to leaf 1, the classified instance (presented at the top of the figure) "travels" bottom-up to the root node. The confidence intervals that are presented in square brackets are used to assess the plausibility of the classification in each inner node. The confidence intervals to which the instance's attributes is assigned, are printed in bold.

1) Alternative Routes

In this case, the instance is within the confidence interval of one or more classes. In this scenario we can assume that there is at least a chance that the instance has been incorrectly classified. Since we concluded that it may belong to a different class, we attempt to determine whether it needs to be "reassigned" to a different descendant node in the tree from what it originally was. For this reason, we assign the instance to all the immediate descendant nodes of the current node (including the originally chosen path) and then produce a weighted average of all the predictions they produce. The weight assigned to each son node is calculated as follows:

$$weight_i = \sum_c (\sqrt{n_c} * \frac{n_c}{T_i})$$

where *i* is the index of the immediate descendant node, *c* is a class in whose confidence interval the classified instance is found, n_c is the number of items from class *c* assigned to this immediate descendant node, and T_i is the total number of training instances in son node *i*. All the weights are then normalized to one, and each prediction is multiplied by this weight and summed. The resulting prediction is returned.

The weight is designed such that the immediate descendant nodes that have a larger proportion of class x will have a higher weight and therefore a higher influence on the final prediction. Note that the threshold *m* (minimum number of instances) ensures that the weights are calculated based on a sufficient set of instances. The reason we use the square root of *n* in the equation instead of *n* itself (as is done in the C4.5 algorithm when the classified instance does not have a value in the split attribute) is our desire to alleviate the problem of imbalanced datasets. We have discovered through experimentation that this method improves performance on imbalanced datasets without affecting performance in balanced ones.

This scenario is shown in Fig. 4. It can be seen that in the root node the use of confidence intervals has shown that the classification may be incorrect; despite being classified as belonging to class *A* by the decision tree, the confidence intervals indicate that the classified instance is more likely to belong to class *C*. Therefore, the following steps are taken:

a) Since the instances of class *C* are assigned to both immediate descendant nodes of the root, both routes will be considered.

b) The left (original) route will produce the following classification: [1,0,0] (100% for class *A*), which was generated in the leaf designated by "1". According to the formula presented above, its weight will be:

$$\sqrt{30} * \frac{30}{230} = 0.714421$$

c) The right (alternative) route will produce the following classification: [0,0,1] (100% for class *C*). According to the formula presented above, its weight will be:

$$\sqrt{70} * \frac{70}{70} = \sqrt{70} = 8.3666$$

d) We now normalize both weights to one, which gives us {0.078672, 0.921328} for the original and alternative paths, respectively.

e) Therefore, the final classification will be [0.078672, 0, 0.921328]. We've gone from providing a wrong classification with absolute certainty to providing the correct classification with a high degree of certainty.

This process is displayed in Fig. 4, with the alternative route shown by the dashed line.

In our experiments we chose to use a higher degree of confidence for the assigned class and another, lower, confidence interval (that is, using a lower certainty level) when attempting to assign the analyzed instance to other classes. The reason for this decision is simple: by using a "smaller" confidence interval for the alternative classes we make it more difficult for instances to be assigned to other classes than to only be "unassigned" from their current class. This way, in cases of lesser certainty we apply the "certainty fines" and in cases of high certainty we use the alternative routes.

2) Certainty fines

In the second case the instance is not within the confidence interval of *any* class. In this scenario the classified instance cannot be assigned with sufficient certainty to any of the classes. Since in this scenario we have no way of knowing to which immediate descendant node(s) to assign the instance, we will instead impose a "certainty fine". This is done by reducing the probability of the leading class (the one with the highest probability) by some value, and dividing it equally among all the classes whose instances were assigned to the current node during the creation of the tree.

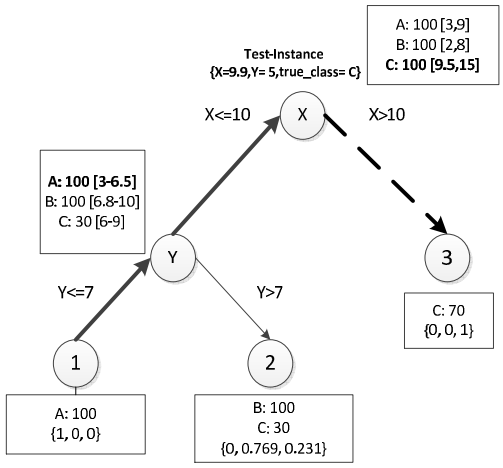

Figure 4. With the confidence interval indicating that class C may be the correct classification, the alternative route (in the dashed line) is also considered.

The idea behind this action is simple. We have a reason to doubt the classification derived by the decision tree, yet we are currently unable to offer an alternative. Therefore, we only somewhat reduce our certainty in the classification, and thus increase the likelihood that the following nodes will be able to change the classification, assuming more "suspicious" attribute values are detected.

The "size" of the fine is calculated using a variation of the Laplace Correction [22]. It is usually used in fields such as text mining [23, 24] in order to prevent probabilities from being zero. We, on the other hand, wish to use the correction in order to reduce large probabilities. For this reason, we define the "fine" as 10% of the largest probability (which is the probability of the assigned class). This way, the larger the current certainty of the classification, the larger the reduction is and vice versa.

An example of this scenario is presented in Fig 5. We have altered the confidence interval of class *C* in the root so that the value 9.9 is no longer inside it. In this case, the following steps will be taken:

a) The certainty by which the classified item is assigned to class *A* (the original classification made by the tree) would be reduced by 0.1 from 1 to 0.9 (1-(1×0.1)=0.9).

b) The 0.1 that was reduced from *A* will be distributed equally among all other classes whose items are in the node (in this case, classes *B* and *C*).

c) The final classification returned by the decision tree would be {0.9, 0.05, 0.05}, thus reducing the classifier's "certainty" of its (mistaken) classification.

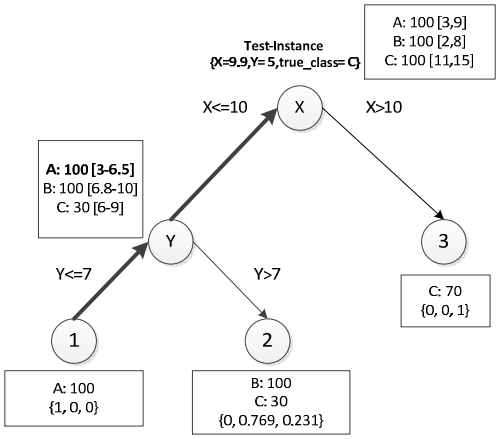

Figure 5. In this scenario no confidence interval in the root contains the value of the classified instance's attribute X. Therefore, a certainty fine will be imposed.

The pseudo code for the classification of an instance using the modified decision tree is presented in Alg. 2. For each node, the algorithm first checks if it is a leaf or not (line 1). If the node is a leaf, the procedure returns the probability of each class, as calculated by the decision tree algorithm used (C4.5, for example). If the node is not a leaf, assign the classified instance to the appropriate immediate descendant node(s) in order to obtain a classification (lines 2-3). Once a classification has been obtained, we check whether the split attribute value of the classified instance is within the confidence interval of the assigned class. If that is the case, then we return the current classification to the parent node (line 4). If it is not the case, we check the confidence intervals of the other classes in the node in order to determine if the classified instance is within the confidence interval of any of them (line 6). If any are found, we apply the *alternative routes* presented in Section 3.2.1 (line 7). If none are found, we use *certainty fines* (line 9).

During the classification phase, the only substantial additional activity is the assignment to alternative routes. Since assigning an instance to an alternative route is equal to the classification of an additional instance, the complexity of classifying an instance cannot exceed *m* – the number of

nodes in the tree; this number will be reached if all attributes are numeric and alternative routes are used at every node (i.e., worst case scenario). Therefore, the computational cost in the worst case scenario will be $O(n \cdot m)$ with n being the number of instances and m the number of nodes in the tree. This is slightly worse than $O(n \cdot h)$ (with h being the height of the tree), the complexity of classifying an instance is the original C4.5 tree, but the complexity remains linear nonetheless.

Algorithm 2 Classify instance

Input: Node *current*_node: a node in the tree
Instance instance: the instance being classified
Output: a set of probabilities, one for each class
1: **IF** Is_Leaf(node)==true **THEN RETURN** node.probs
2: Probs ← Classify_Instance(node.Get_Relevant_descendant(), instance)
3: Current_classification_id ← Get_Current_Classification(probs)
4: **IF** (Instance_Is_Within_Confidence_Interval(node.attribute_averages, node.stdevs, Current_classification, instance) **THEN RETURN** Probs;
5: **ELSE**
6: alternative_class_ids ← Get_Class_ID_Whose_Stdevs_Contain_Instance(node.attribute_averages, node.stdevs, instance)
7. **IF** (alternative_class_ids.Length > 0) **THEN RETURN** Get_Weighted_descendant_Prediction(alternative_class_ids, node.descendants, instance)
8. **ELSE**
9. **RETURN** Impose_Certainty_Fine(probs, node.num_of_instances)

3) Dealing with Missing Values

In our experiments, we have learned that missing values pose a problem when using confidence intervals. Due to the C4.5 algorithm's method of assigning instances with missing values to all possible paths, such instances were targets for changes along their many paths in the tree. This "excess diversity" actually harmed the classification outcome and therefore we modified the algorithm so that the proposed method would not be used in nodes following those for which there were missing values.

For example, if a classification path in a decision tree consists of six nodes and a classified instance is missing a value in the attribute needed for the third node (from the tree root), then the proposed method will not be applied in nodes four and five (the method will not be used in node six because it is a leaf). The proposed method will be used in node one, the root, and two.

IV. EVALUATION

This section is divided into two parts. In the first, we evaluate the model's performance on both binary and multi-class datasets, use statistical tests in order to determine which version performs best and analyze the percentage of items affected by the proposed method. In the second half of this section we evaluate the proposed method's performance on varying training set sizes.

A. *Analysis of the proposed method's performance on binary and multi-class datasets*

The proposed method was tested on 10 two-class datasets (binary problems) and 7 multi-class datasets in order to assess its contribution (see Table 1). All datasets are well known and available online (from the UCI[1] repository). On each dataset we tested the following four decision trees: original C4.5 algorithm (Org), the proposed ConfDTree when using the normal distribution for deriving the confidence intervals ($\text{ConfDTree}_{\text{NORM}}$), the proposed ConfDTree when using the *t*-distribution for deriving the confidence intervals ($\text{ConfDTree}_{\text{TDIST}}$), and the proposed ConfDTree when using both the normal distribution and *t*-distribution for deriving the confidence intervals ($\text{ConfDTree}_{\text{COMB}}$). We also tested and applied the RandomForest classifier, which we used as an upper bound, in order to compare it with the ConfDTree and understand the potential improvement of the decision tree.

For the comparison of the performance of the four decision trees and RandomForest we chose to use the AUC (area under the ROC curve) measure [25]. For the two-class (binary) datasets this is a straightforward and well accepted comparison measure. For the multi-class datasets we calculated the AUC for each class by defining that class as "positive" and all other classes as "negative", and deriving the following measures as suggested in [26]:

1. *average AUC* – a simple average of all calculated AUCs
2. *weighted average AUC* – each AUC was assigned with a weight that was equal to the percentage of its "positive" class of the total number of instances (thus giving more weight to the more common classes).

We used both balanced and unbalanced datasets in order to obtain a better insight on for which circumstances the proposed method contributes to the classic decision tree algorithm. The degree of imbalance of each dataset is presented in Table 1. The proposed method was implemented on the open source machine learning platform Weka [27] and all experiments were run on it. All results were obtained using a 10-fold cross validation.

The experiments were run with following settings:

1. The confidence interval that was used in order to determine whether an attribute is inside the confidence interval of the assigned class was that of two standard deviations for the normal distribution or a confidence level of 0.995 (i.e., α=0.5%) for the t_value distribution (line 4 in Alg. 2).
2. The confidence interval that was used in order to determine whether an attribute is inside that of other classes (line 6 in Alg. 2) was generated using one standard deviation for the normal distribution or a confidence level of 0.9 (i.e., α=10%) for the t_value distribution.
3. The value of *m*, the minimum number of instances for which confidence interval is computed was set to 5.

[1] http://archive.ics.uci.edu/ml/

TABLE I. DATASETS PROPERTIES AND THE IMPROVEMENT OF AUC OF THE PROPOSED METHOD.

Name	Num of Classes	Num of Numeric Atts	Num of Instances	Imbalance Ratio	ConfDTree [NORM]	ConfDTree [TDIST]	ConfDTree [COMB]	Random Forest
cancer	2	30/30	569	1 : 1.168	3.1%	3.1%	3.6%	6.9%
contraceptive	2	6/9	1473	1 : 3.423	1.3%	4.2%	3.5%	21.7%
credit	2	6/15	690	1 : 1.247	0.3%	0.7%	0.6%	8.5%
diabetes	2	8/8	768	1 : 1.865	4.5%	8.7%	8.2%	11.2%
ecoli	2	7/7	336	1 : 8.6	3.2%	2.5%	3.4%	17.5%
ionosphere	2	34/34	351	1 : 9.14	2.0%	8.7%	8.9%	16.7%
pima	2	8/8	768	1 : 1.865	3.3%	6.0%	6.0%	5.4%
spam	2	57/57	4601	1 : 1.5377	1.8%	-6.9%	2.9%	6.1%
yeast	2	8/8	1004	1 : 9.14	0.0%	7.8%	0.0%	27.2%
MiniBooNE	2	50/50	130,000	1 : 2.5	6.0%	-33.4%	8.1%	12.5%
Autos	6	7/17	205	1 : 7.3 : 9 : 10.6 : 18 : 22.3	0.09% (0.10%)	2.67% (3.04%)	2.67% (3.04%)	4.89% (5.25%)
Glass	6	9/9	214	1 : 1.44 : 1.88 : 9.66 : 7.77 : 8.44	0.74% (1.42%)	4.57% (3.83%)	4.44% (3.85%)	13.19% (11.15%)
Letter	26	16/16	20,000	max ratio of 1 : 1.1	0.04% (0.04%)	6.09% (6.07%)	6.11% (6.08%)	8.35% (8.33%)
Segment	7	19/19	2310	1 : 1 : 1 : 1 : 1 : 1 : 1	0.46% (0.46%)	1.24% (1.24%)	1.08% (1.08%)	1.95% (1.95%)
Vehicle	4	18/18	846	1 : 1.065 : 1.094 : 1.095	2.92% (2.98%)	8.34% (8.51%)	8.07% (8.18%)	14.35% (14.52%)
Vowel	7	10/13	989	1 : 2 : 2 : 2 : 2 : 2 : 2	1.47% (1.18%)	4.83% (4.5%)	4.61% (4.22%)	11.56% (10.77%)
Waveform	3	40/40	5000	1 : 1 : 1	11.17% (11.18%)	12.69% (12.7%)	12.69% (12.80%)	21.31% (21.33%)

Datasets properties include: the number of classes, number of instances and the ratio between the majority and minority classes.
The improvement in AUC, realtive to the original C4.5. algorithm, is presented for the three variations of the proposed method and for all datasets. Weighted average AUC is presented for multi-class datasets in parentheses. RandomForest is also presented, as an upper-bound.

Results for the three versions of the proposed method ($ConfDTree_{NORM}$, $ConfDTree_{TDIST}$, and $ConfDTree_{COMB}$) are presented in Table 1. As can be seen in Table 1, all variations of the proposed method outperform the original C4.5 algorithm (J48 in Weka) except for the $ConfDTree_{TDIST}$ on the datasets spam and MiniBooNE. The $ConfDTree_{COMB}$ clearly performs best – it achieved high scores on all of the binary and multi-class datasets (Table 1).

Using both a paired t-test and the Wilcoxon test we were able to determine that the proposed method outperforms the original decision tree with p-value<0.05. From Table 2 we can see that according to both tests the average AUC and weighted average AUC of the $ConfDTree_{TDIST}$ and $ConfDTree_{COMB}$ were significantly better than the original version of the decision tree for the multi-class datasets. For the binary datasets both the $ConfDTree_{COMB}$ and $ConfDTree_{NORM}$ outperformed the original version of the decision tree. To conclude, the combination of t-distribution and normal distribution version of the ConfDTree is the preferable choice according to our experiments.

Since the proposed method is designed to tackle a specific problem, namely, of attributes whose values are "uncharacteristic" of their class, it would be impossible to understand the results without knowing what percentage of instances has been affected by the method. We decided to define "affected" as instances whose assignment probabilities were modified by 10% or more. We chose this definition because we believe that other definitions (for example, instances whose assigned class was changed) were inadequate since they would not represent the many instances in which the proposed method had a serious effect on the classification. One such example could be the reduction of the certainty of a mistaken classification from 100% to 51%. The results of the analysis are presented in Table 3.

It is clear that when using the t-value distribution for the generation of the confidence intervals, the percentage of affected instances is higher. This is due to the fact that this method can be applied on all nodes whose split attributes is numeric (without the need for checking whether the split attribute is normally distributed for each class). However, to our surprise, we were not able to find correlation between the percentage of affected instances and the performance of the proposed method.

B. Evaluation of the proposed method using varying training set sizes

We hypothesized that the method proposed here would be especially beneficial when applied on small training sets. Our rationale was that while small training sets might make it more difficult for the C4.5 to generate accurate split values for many attributes, the proposed method will be able to (at least partially) compensate for incorrect classifications by using confidence intervals to identify them.

We tested this hypothesis by generating different sizes of training sets for all the algorithms presented in the previous section. By generating training set sizes varying from 10% to 90% of the dataset, we hoped to identify clear trends in the proposed method's performance. For each training set size, we randomly created 10 divisions (while maintaining the

imbalance ratio) of the dataset and averaged the model's performance. For the multi-class datasets we present the comparison of their average-AUCs.

TABLE II. PAIR-T TEST AND WILCOXON TEST RESULTS

Measure	Dataset	Decision tree	Wilcoxon	Paired-*t*
AVG	Multi-class	Org vs. ConfDTree$_{NORM}$	z= -2.023 p =0.043	t= -1.650 p =0.15
AVG	Multi-class	Org vs. ConfDTree$_{TDIST}$	z = -2.366 p =0.018	t = -4.429 p =0.004
AVG	Multi-class	Org vs. ConfDTree$_{COMB}$	z = -2.366 p =0.018	t =-4.332 p =0.005
Weighted AVG	Multi-class	Org vs. ConfDTree$_{NORM}$	z = -2.366 p =0.018	t =-1.716 p =0.137
Weighted AVG	Multi-class	Org vs. ConfDTree$_{TDIST}$	z = -2.366 p =0.018	t =-4.378 p =0.005
Weighted AVG	Multi-class	Org vs. ConfDTree$_{COMB}$	z = -2.366 p =0.018	t =-4.251 p =0.005
AVG	Binary	Org vs. ConfDTree$_{NORM}$	z =-2.803 p =0.005	t =-4.182 p =0.002
AVG	Binary	Org vs. ConfDTree$_{TDIST}$	z =-0.968 p =0.333	t =0.106 p =0.075
AVG	Binary	Org vs. ConfDTree$_{COMB}$	z =-2.803 p =0.005	t =-4.385 p =0.002

TABLE III. PERCENTAGE OF INSTANCES AFFECTED BY THE PROPOSED METHODS IN THE BINARY-CLASS AND MULTI-CLASS DATASETS

Dataset	ConfDTree [NORM]	ConfDTree [TDIST]	ConfDTree [COMB]
Cancer	5.80%	70.10%	6.6%
Contraceptive	2.1%	44.20%	12.5%
Credit	1.10%	3.50%	2.2%
Diabetes	4%	78.40%	15%
Ecoli	3.90%	56%	7.2%
Ionosphere	49.40%	83%	81.7%
Pima	5.50%	84.30%	14.2%
Spam	10.10%	99%	16.6%
Yeast	1.20%	7.90%	0.7%
MiniBooNE	14.3%	99%	29.4%
Autos	11.3%	18.2%	17.1%
Glass	1.1%	12.7%	5%
Letter	11.8%	74.4%	73.89%
Segment	3.8%	87%	76.3%
Vehicle	10.4%	78.7%	47.8%
Vowel	21%	71.4%	40.7%
Waveform	16.8%	99%	36.9%

The results of the comparison are presented in Figure 6, which shows the *average relative improvement* for the binary and multi-class datasets. It is clear that there is a downward trend in the relative improvement for both types of datasets as the size of the training set increases.

We found that in the binary datasets, the downward trend in relative improvement was best observed in three datasets – *Ecoli*, *Ionosphere* and *Yeast*. These results are very interesting, as they are the three most imbalanced datasets, and Ecoli and Ionosphere are also the two smallest datasets. Moreover, the three datasets in which there was virtually no decline in relative improvement, *Credit*, *Cancer* and *MiniBooNe*, are also the most balanced (see Table 1). This leads us to conclude that the proposed method is especially beneficial in cases where the available training set is small or highly imbalanced.

In the multi-class datasets, the downward trend was clear for 4 out of the 7 datasets (*Letter, Vowel, Segment* and *Vehicle*) and a (relatively) fixed improvement existed for 2 additional datasets (*Autos* and *Waveform*). For the 7th dataset (*Glass*) there was an upward trend which was later reversed to a downward trend.

When attempting to understand these results, three factors need to be considered – the size of the dataset, the number of classes and the degree of imbalance. We found that two of the three datasets in which there was no downward trend (*Autos* and *Glass*) have a small number of instances (around 200), a large number of classes (6) and the highest degree of imbalance of all datasets. It seems logical to assume that this combination makes it very difficult for reliable confidence intervals to be created for these datasets with very few samples. In the case of the third dataset without a downward trend (*Waveform*) the case is exactly the opposite; it has a relatively large number of instances (5000), only 3 classes and no imbalance. We believe this indicates that even a small percentage of the dataset was sufficient for an accurate model to be created.

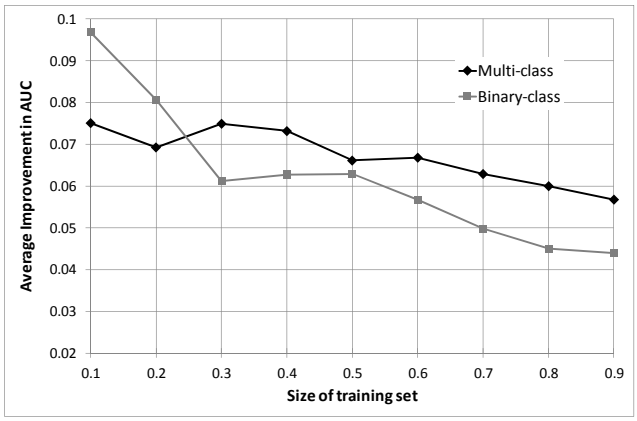

Figure 6. The relative improved (averged over all multi-class and binary-class datasets) of the proposed method over the original C4.5 as a function of the percentage of the training set of all instances.

V. DISCUSSION

In this paper we present and evaluate a method for enhancing decision trees. The method can be used to deal with three important problems of decision trees: reduced performance when applied on small training sets, the rigidness of the classification process and outlier attribute values that interfere with correctly classifying an instance.

Evaluation results show that the proposed method performs significantly better than the original C4.5 algorithm, both for binary and multi-class datasets. The improvement was even larger when the size of the training sets was reduced (from an average 5% improvement to 9% on the smallest training sets), a testament to the proposed method's robustness. In addition, the results show that the more imbalanced or diverse the dataset, the greater the improvement the proposed method is likely to yield.

The fact that the proposed method performs better on imbalanced and diverse datasets is not surprising. In imbalanced datasets it is less likely that the underrepresented

class will be correctly classified, while in multi-class datasets the decision process is prolonged because of the need to use binary splits to classify multiple examples.

The proposed method's ability to be integrated with every decision tree algorithm is important. This enables the most suitable algorithm to be selected for a specific dataset and still have the benefit of using confidence intervals.

The proposed algorithm has also two drawbacks: (1) it slightly increases the computational cost of classifying a new instance; and (2) it reduces the comprehensibility of the model. In particular some instances are affected by our method and eventually assigned a different class distribution. This also indicates that a simple transformation of the decision tree to a set of if-then rules is not as simple.

We believe, however, that the loss of comprehensibility of the model is minimal since most of classified instances will not be affected by the proposed model, as we only target "outliers". For the classification of outliers, "certainty fines" (whose meaning is clear) can be easily annotated on the path shown to the user. Only the alternative routes may actually reduce comprehensibility; however they also can be marked to indicate that additional paths were inspected.

Four future research directions are currently being considered. The first direction regards the issue of missing values. Currently, we do not apply the method beyond the point in which such values are detected. We are considering using the values of the instance's other normally distributed attributes in order to determine the paths to which it should be classified and their relative weight. The second research direction deals with the issue of imbalance. The proposed method already performs better on imbalanced datasets, but we would like to add additional improvements such as taking the relative imbalance into account when choosing alternative routes and certainty fines. The third direction is an attempt to use the method in conjuncture with ensemble algorithms which utilize decision trees (e.g., RotationForest and RandomForest). We will try to determine in which scenarios the use of the method improves results and what modifications (if any) are required for the algorithm. The final research direction currently being considered is the expansion of the method to nominal values. We wish to try and adapt the method so that it can infer whether an item's nominal attributes are unlikely when considering its assigned class. Techniques similar to those used by SMOTE [28] and similar algorithms will also be considered.

REFERENCES

[1] Rokach, L. and O. Maimon, Data mining with decision trees: theory and applications. World Scientific Pub-lishing, Singapore, 2008.

[2] Quinlan, J.R., C4. 5: programs for machine learning. Vol. 1. 1993: Morgan kaufmann.

[3] Chawla, N.V., N. Japkowicz, and A. Kotcz, Editorial: special issue on learning from imbalanced data sets. SIGKDD Explor. Newsl., 2004. 6(1): p. 1-6.

[4] Provost, F. and P. Domingos, Well-Trained PETs: Improving Probability Estimation Trees. Technical Report CDER #00-04-IS, 2001.

[5] Lin, H.-Y., Efficient classifiers for multi-class classification problems. Decision Support Systems, 2012. 53(3): p. 473-481.

[6] Breiman, L., Random Forests. Machine Learning, 2001. 45(1): p. 5-32.

[7] Quinlan, J.R., Induction of decision trees. Machine Learning, 1986. 1(1): p. 81-106.

[8] Quinlan, J.R., C4.5: Programs for Machine Learning. 1993.

[9] Breiman, L., et al., Classification and regression trees. Belmont: Wadsworth, 1984.

[10] Breiman, L., Technical Note: Some Properties of Splitting Criteria. Machine Learning, 1996. 24(1): p. 41-47.

[11] Rodrıguez, J.J., L.I. Kuncheva, and C.J. Alonso, Rotation forest: A new classifier ensemble method. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2006. 28(10): p. 1619-1630.

[12] John, G.H., Robust Decision Trees: Removing Outliers from Databases. Proceedings of the First International Conference on Knowledge Discovery and Data Mining, 1995: p. 174-179.

[13] Last, M., O. Maimon, and E. Minkov, Improving Stability of Decision Trees. International Journal of Pattern Recognition and Artificial Intelligence,, 2002: p. 145-159.

[14] Zadrozny, B. and C. Elkan, Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. Proceedings of the Eighteenth International Conference on Machine Learning, 2001.

[15] Chm-les X, L.C., O. CA, and R.J. Yan, Decision Tree with Better Ranking.

[16] Mccallum, R.A., Instance-Based Utile Distinctions for Reinforcement Learning with Hidden StateR. Andrew Mccallum Proceedings of the Twelfth International Conference on Machine Learning, 1995: p. 387-395.

[17] Massey, F.J., The Kolmogorov-Smirnov Test for Goodness of Fit. Journal of the American Statistical Association, 1951. 46(253): p. 68-78.

[18] Janikow, C.Z., Fuzzy decision trees: issues and methods Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 1998. 28(1): p. 1-14.

[19] Olaru, C. and L. Wehenkel, A complete fuzzy decision tree technique. Fuzzy Sets and Systems, 2003. 138(2): p. 221-254.

[20] Zadorny, B. and C. Elkan, Obtaining calibrated probability estimates from decision trees and naive Bayesian classiers. In: Proc. Eighteenth Internat. Conf. on Machine Learning, 2001: p. 609-616.

[21] Esposito, F., D. Malerba, and G. Semeraro, A comparative analysis of methods for pruning decision trees. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 1997. 19(5): p. 476-491.

[22] Kohavi, R., B. Becker, and S. D, Improving Simple Bayes. The 9th European Conference on Machine Learning, 1997: p. 78-87.

[23] Ponte, J.M. and W.B. Croft, A language modeling approach to information retrieval, in Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval1998, ACM: Melbourne, Australia. p. 275-281.

[24] Lafferty, J. and C. Zhai, Document language models, query models, and risk minimization for information retrieval, in Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval2001, ACM: New Orleans, Louisiana, United States. p. 111-119.

[25] Demšar, J. and ar, Statistical Comparisons of Classifiers over Multiple Data Sets. J. Mach. Learn. Res., 2006. 7: p. 1-30.

[26] Hand, D.J. and R.J. Till, A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. Machine Learning, 2001. 45(2): p. 171-186.

[27] Hall, M., et al., The WEKA Data Mining Software: An Update. SIGKDD Explorations, 2009. 11(1).

[28] Bowyer, K.W., et al., SMOTE: Synthetic Minority Over-sampling Technique. Journal Of Artificial Intelligence Research, 2002. 16: p. 321-357.