Analysing the Termination of Term Rewriting Systems using Data Mining

J. Piris
DSIC, Universitat Politècnica de València,
Camí de Vera s/n, 46022 València, Spain
jpiris@dsic.upv.es

H. Fabregat
Universitat Politècnica de València,
Camí de Vera s/n, 46022 València, Spain
herfabma@inf.upv.es

M.J. Ramírez-Quintana
DSIC, Universitat Politècnica de València,
Camí de Vera s/n, 46022 València, Spain
mramirez@dsic.upv.es

During the last decades, researchers in the field of Term Rewriting System (TRS) have devoted a lot of effort in order to develop techniques and methods able to demonstrate the termination property of a TRS. As a consequence, some of the proposed techniques have been implemented and several termination tools have been developed in order to automatize the termination proofs. From 2004, the annual Termination Competition is the foro in which research groups compare their tools trying to provide termination proofs of as many TRS as possible. This event generates a large amount of information (results obtained by the different tools, time spent on each proof, ...) that is recorded in databases. In this paper, we propose an alternative approach to study the termination of TRS: to use data mining techniques that, based on the historical information collected in the competition, generate models to explore the termination of a TRS. The goal of our study is not to develop a termination tool but to show, for the first time, what machine learning techniques can offer to the analysis of TRS termination.

1 Introduction

Termination is a fundamental topic in computer science for its important practical implications. The problem of program termination is defined as: *to determine using only a finite amount of time whatever a given program will always finish running or could execute forever, or return the answer "unknown'*. This problem, also known as the *halting problem*, was proved to be undecidable by Turing [23]. Nevertheless, it have been shown that, under some conditions, it is possible to check program termination.

Thus, in 1970 Manna & Ness [21] proposed a criterion for proving termination of functional programs based on *reduction orderings*. Since then, many other techniques based on different formalisms have been proposed to prove termination [6] [20] [7]. For instance, the recursive path orders, the polynomial interpretations or the Knuth-Bendix orders, to mention some of them. In 1997, Arts & Giesl [3] proposed the *Dependency Pair* criterion that allows to prove the termination of more classes of programs than previous techniques. This fact encouraged researchers to continue working in this field, not only defining new techniques but also developing automated termination tools, such as AProVe [14] [13] (`http://aprove.informatik.rwth-aachen.de/`), Mu-Term [1] (`http://zenon.dsic.upv.es/muterm/`) or Julia [2] (`http://julia.scienze.univr.it/`).

In such a context, since 2004 a termination competition takes place every year co-located with some other related events [22]. The objective of the competition is to compare the different termination tools that participate in each problem category. The result of the competition is a ranking of the systems (per

Submitted to:
PROLE 2015

category) generated by taking into account the number of programs successfully analysed and the time devoted in each proof. But more interestingly, this competition provides a source of data with thousands of programs analysed. All this information (the collection of programs used in each edition, and other data about the proofs performed by the tools) is recorded in databases that are available at the web page of the competition (`http://termination-portal.org/wiki/Home`). In particular, in this work we use the final result obtained by the tools: the TRS is terminating, non terminating or its termination is unknown (the tools have not been able to provide a proof).

In this paper we propose to address the termination problem by analysing the data of the termination competition. The analysis of large amounts of data is a well-known problem in the area of data bases. In fact, the term Data Mining [16] refers to the subfield of computer science that provides methods and techniques for analyzing large sets of data. Data mining is one of the phases of a more general process known as *Knowledge Discovery in Databases*, KDD [9]. KDD is defined as the process of discovering useful knowledge from a collection of data. This process consists of a sequence of iterative phases:

- Problem definition: identify the goal of the KDD process.
- Data selection: select the data samples to be used for knowledge discovery.
- Data preparation: clean wrong data, handle missing values and transform data.
- Data mining: apply data analysis and machine learning techniques to discover hidden patterns (or models).
- Interpretation: determine whether or not the pattern is interesting.
- Deployment: incorporate the knowledge into another system for further application.

Therefore, the final purpose of the KDD process is to use the discovered knowledge to support the decision making whereas the goal of data mining (the most distinguishing stage of the KDD) is to extract new knowledge from data.

Data mining involves two kinds of tasks: predictive and descriptive. In predictive tasks, the goal is to estimate (to predict) for an instance (or database register) the value of an attribute (or field) of interest (called the class attribute or the target variable) by using the other attributes (called the predictive attributes or independent variables). A predictive task is called classification when the target attribute is nominal and regression when the class attribute is numerical. In descriptive tasks, the goal is to explain, summarise or group the data. Examples of descriptive tasks are clustering and association rules learning.

The KDD process in general, and the data mining in particular, have been applied with success to different application areas, such as bussines (marketing, retail data analysis, stock selection,...), astronomy, biology, medicine, banking, ... Given that the termination competition provides us with historical data, a logical question that arises is: can data mining techniques be applied to analyse program termination? To answer this question, in this paper we apply data mining techniques to learn models able to predict the termination of a TRS. In order to do this we start with the data collected in the termination competition. It has been already established that the good (or bad) results after the data mining process depends to a large extent on an appropriated data selection and preparation, so our first task is to inspect the raw data and determine which information we are going to use and how we should represent the TRS to make possible the application of data mining techniques. After that, we carry out some experiments using different data mining techniques. Taking a look at the collected data, it can be realized that the majority of TRS are terminating. This means that we have a lot of instances (TRS) belonging to the class 'terminating', and very few instances of the other two classes ('non-terminanting' and 'unknown'). Following the machine learning terminology, we say that the data are imbalanced. This is a problem prevalent in many applications, for instance fraud/intrusion detection or medical diagnosis. In such cases, standard classifiers tend

to be overwhelmed by the large classes and ignore the small ones. This could be a drawback when the interest is focused on the minority classes (for instance, in the diagnosis of a rare disease). This suggests us two experimental scenarios we firstly explore in this paper: (1) to learn a classifier by directly using the original (imbalanced) dataset and (2) to use well-known techniques specially defined for learning with imbalanced data [15]. In both cases, we tackle the problem as a multi-class classification one, in that all the instances are labeled with one of the three possible class values mentioned above[1]. But, an alternative perspective would be to consider that we are dealing with a binary problem (with only two classes, 'terminating' and 'non-terminanting') and that there are some instances (those originally labeled as 'unknown') that have not class label assigned[2]. We experimentally study this latter scenario too.

It is worth mentioning that the main goal of this paper is to present a first attempt for facing the termination of TRS from a different point of view. Up to our knowledge, the idea of using data mining to study and analyze termination of programs is new and there is not precedent about it. Our long-term insight is that an analysis based on data-mining could be used by the termination tools in order to make better decisions when performing their proofs.

The paper is organised as follows. Section 2 relates the process followed for selecting the data from the termination competition database, defines the attributes we are going to use for describing each TRS and the construction of the dataset for the analysis. Section 3 presents the experiments performed and discusses the results obtained. Finally, Section 4 concludes the paper and outlines the future work.

2 Data extraction and processing

In this section we describe the construction of the dataset D to be used in Section 3. Each row in D represents a TRS whereas the columns represent their attributes (fields). Therefore, each TRS R can be denoted as a tuple $R = \langle x_1, x_2, \ldots, x_n, y \rangle$ where x_i are the predictive attributes and y is the class attribute.

The Termination Competition portal (`http://www.termination-portal.org`) contains, among other informations, a historical record of the results of the competitions from 2004 to 2014, and the Termination Problems Database, TPDB (a collection of all problems used in the competitions). In each edition, the competition is organised in several categories. Table 1 shows all the categories involving Term Rewriting Systems.

`TRS Standard`
`TRS Relative`
`TRS Innermost`
`TRS Equational`
`TRS Contextsensitive`
`TRS Outermost`
`TRS Standard Certifying`
`TRS Conditional`
`TRS Relative Certifying`

Table 1: Termination competition categories for Term Rewriting Systems.

In our study we only use the data corresponding to the competitions of TRS from 2008 to 2013, in which the rewriting systems are represented in an xml-format and the results in csv-format. As having a

[1] When learning takes place using a dataset such that all the instances are labeled, the learning is called supervised.

[2] When learning takes place using a dataset such that some instances, but not all, are unlabeled, the learning is called semi-supervised.

uniform format makes easier data processing and taking into account that the selected competition period contains sufficient data for our purposes, we do not use data from the competitions held before 2008.

The first field in the result file contains a reference to the path where the TRS is placed in the TPDB. The rest of the fields contain the results of the proofs performed by the tools: YES when the tool has proved that the TRS is terminating, NO when it has proved that the TRS is non terminating and $MAYBE$ if the prover has not been able to find any proof. A $TIMEOUT$ result can also be produced when the process is halted because the proof has exceeded a previously fixed time limit. There are few TRS for which all provers return a $TIMEOUT$. In fact, in most cases, at least one of the proofs ends returning any of the other three outcomes. As explained below, these three values are used to label the TRS. This is the reason why $TIMEOUT$ values are not considered for this study.

The next step is to assign a unique class label y ($YES, NO, MAYBE$) to every TRS. Given that not all the tools are able to complete the termination proofs, for each TRS we have to decide the true class label regarding all of its proof results. This process is carried out by applying the following procedure:

1. If at least one tool has proved the termination of the TRS and no other tools have said NO, then $y = YES$.
2. If at least one tool has disproved the termination of the TRS and all other tools have said nothing or $MAYBE$, then $y = NO$
3. If there is no tool that has proved or disproved the termination of the TRS but at least one of them has said $MAYBE$, then $y = MAYBE$.
4. If there is a tool proving termination and another disproving it for the same TRS, then the TRS is discarded (it is not included in D).

Note that the last rule allows us to detect the inconsistencies (there are quite a few cases mainly due to mistakes in the proofs or bugs in the implementation of the tools) and to clean the dataset. The rules are applied in order to guarantee that a unique class label is produced.

At this point, all that remains is to define the predictive attributes x_i. Since a TRS R is described through its attributes, one option is to think about x_i in terms of properties of R. Of course, we can only consider syntactic properties that can be defined regarding the set of rules in R.

In that follows, we use the standard notation in the term rewriting literature [4]. A signature Σ is a finite set of function symbols with their arity. Function symbols of arity 0 are called constants. Let V be a set of variables. By $T(\Sigma, V)$ we denote the set of terms over $\Sigma \cup V$. A *rewriting rule* is an ordered pair of terms denoted as $l \to r$, where l is not a variable, and every variable occurring in r also occurs in l. Rules of the form $C \Rightarrow l \to r$, where C is a set of equations such that any variable occurring in C also occurs in l, are called conditional. A (conditional) TRS is a finite set of (conditional) rewriting rules. The set of properties that conform the description (attributes) of a term rewriting system R is:

- *Size*: the cardinality of Σ.
- *Maximun height*: defined as $Maximun\_height(R) = max\{max\{height(l), height(r)\} | l \to r \in R\}$ where the height of a term t is defined as:

$$height(t) = \begin{cases} 0 & t \text{ is a constant or a variable} \\ 1 + max\{height(t_1), ..., height(t_m)\} & t = f(t_1, \ldots, t_m) \end{cases}$$

- *Linearity*: A term t is linear if no variable occurs more than once in it. R is linear if for every rule $l \to r \in R$, l and r are linear.

- *Left/Right linearity*: R is left/right linear if for all rules $l \to r \in R$, the term l/r is linear.
- *Collapse*: R is collapsing, if in R there exists at least a rule of the form $l \to x$, with $x \in V$.
- *Variable duplication*: R is duplicating if at least for a rule $l \to r \in R$ there is a variable that occurs more times in r than in l.
- *Ground*: R is ground if any of its rules contain variables.
- *Left ground/Right ground*: When for every rule $l \to r \in R$, the term l/r do not contains variables.
- *Shallow*: R is shallow if all variables occur at depth at most one in each term from the two sides of each rule. The depth of the variables $v \in V$ ocurring in a term t is defined as follows:

$$depth(v,t) = \begin{cases} 0 & v = t \\ 1 + max\{depth(v,t_1),...,depth(v,t_m)\} & t = f(t_1,\ldots,t_m) \end{cases}$$

- *Recursive*. We use a basic notion of recursive. A rule $l \to r \in R$, is recursive if the most external symbol of l appears in r. R is recursive if it contains at least a recursive rule.

The above mentioned properties were implemented in Haskell. As the TRS downloaded from TPDB are in an xml-format, in order to compute the value of the properties with the Haskell programs we have transformed the notation of the TRS to a suitable format[3] by using a tool provided in the *termination portal*.

The final step is to join properties and class labels in a unique table. This dataset D has the right format to be analised by data mining techniques. Figure 1 graphically illustrates the whole process of the dataset generation.

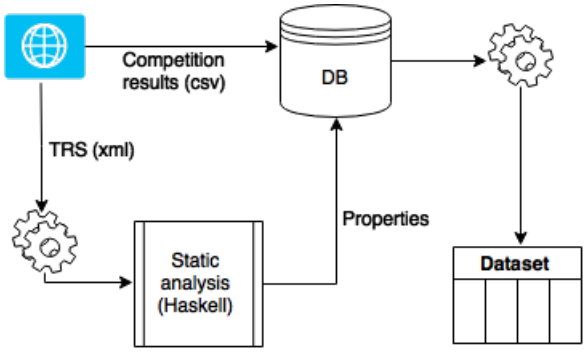

Figure 1: Process for selecting and pre-processing the data of the termination competition.

3 Experimental Study

In this section we aim at establishing whether data mining models give good predictions for the TRS termination problem. To do this, we study the behaviour of four classification methods using the RWeka [17] and the Caret [18] libraries from R. Concretely, we use the following techniques: a decision tree

[3]Presented at the 1st International Workshop on Haskell and Rewriting Techniques (HART 2013) - available online at: http://www.jaist.ac.jp/ c-sterna/publications/Felgenhauer-Avanzini-Sternagel-HART13.pdf

method (J48), a support vector machine with Linear Kernel (svmLinear), a neural network (nnet) and k-Nearest Neighbors (IBk). Table 2 shows the acronym used for denoting the techniques in the tables of results.

Acronym	Technique
DT	Decision Tree
SVM	Support Vector Machine
NN	Neural Network
kNN	Nearest Neighbours

Table 2: Four different learning techniques that are considered for the experiments.

As we have collected the TRS used in several editions of the termination competition, we have enough data to apply a typical model construction procedure which consists in splitting the dataset D into two disjointed sets, D_{Train} and D_{Test}, and using D_{Train} for training the classifiers and D_{Test} for evaluate the performance of the generated models. In this way, the classifier evaluation is more fair because it is done by using a set of data different to the data used for generating the model. For the experiments we have randomly split D into 70% train and 30% test but preserving the overall class distribution of the data.

In the first experiment conducted we used two evaluation metrics [10]: *accuracy* and the *Cohen's Kappa coefficient*. The accuracy is a measure of how close to the true values are the predictions given by a classifier. So, the accuracy is calculated as the proportion of instances in D_{Test} that are successfully classified. It is the most commonly used metric for assessing the performance of classifiers. On the other hand, the Cohen's Kappa coefficient measures the degree of randomness in the predictions. Cohen's Kappa coefficient ranges from 0 (random classification) to 1. It is considered a more robust measure than accuracy, since it takes into account the agreement occurring by chance.

Table 3 shows the performance of the four selected techniques. As can be seen, all the models behave poorly from a predictive point of view. The model with the best accuracy is the decision tree whereas the kNN gets the best value for the kappa measure.

Technique	Acc	Kappa
DT	0.7248	0.2494
SVM	0.7264	0.1681
NN	0.7201	0.2549
kNN	0.6560	0.2919

Table 3: Performance of different classification techniques for the dataset of the TRS termination competition.

In order to investigate the reasons why the techniques have that behaviour, we study the classification errors for each class. This can be done by analysing the confusion matrix, a table whose columns represent the instances in a predicted class, while each row represents the instances in an actual class (the true class label). The confusion matrix for the decision tree is as follows:

```
=== Confusion Matrix ===
   Predicted
    a    b    c          Actual
    4    6   42 |    a = MAYBE
    6   37   99 |    b = NO
    3   16  412 |    c = YES
```

As we have mentioned in Section 1, the majority of TRSs have YES as actual class (the $88,4\%$ of the instances). Regarding the confusion matrix, it can be also seen that the errors are mainly associated to the minority classes (approximately, the 74% of instances of class NO and the 88% of instances of class $MAYBE$ are missclassified). The confusion matrices for the other classifiers show the same facts. All of this seems to confirm that the imbalance of classes could be negative affecting the performance of the classifiers.

If the latter reasoning is right, then the accuracy is not a suitable evaluation measure to use because it does not take into account the class distributions. To illustrate this fact let us considering, for instance, an imbalanced binary problem with two classes (T and F), such that the 90% of instances are of class T and the 10% of class F. A classifier that always predicts class T gets a good accuracy (90%) but it incorrectly classifies all the instances of class F (the minority class). It has been shown that for imbalanced problems a better measure than accuracy is the *Area under the ROC curve* (AUC) [11]. A ROC curve [8] of a binary classifier (in that the classes are referred as positive and negative, respectively) depicts relative trade-offs between benefits (the rate of positive instances correctly classified or true positive rate) and costs (the rate of negative instances incorrectly classified as positives, or false positive rate). To compare classifiers a common method is to calculate the Area Under the ROC curve (AUC)[4]. Hence, in the following experiments we are going to use the AUC measure.

Additionally, to deal with classification of imbalanced datasets, we can also work at the data level by running re-sampling techniques that try to diminish the effect caused by the class imbalance [12]. The different re-sampling techniques can be grouped in two categories:

- *Under-sampling*: to randomly select a subset of instances of the majority class.
- *Over-sampling*: to increase the number of instances of the minority classes (generating artificial instances or repeating the existing ones).

In [19], the authors experimentally analysed a wide collection of re-sampling methods. According to their results, they recommended to use the $SMOTE$ method [5] (an over-sampling technique).

Table 4 shows the results obtained by applying the over-sampling method $SMOTE$ to the training dataset D_{Train} before the learning stage. We have also included the results for the four classification methods learnt from the original data as reference. As we can see, in general the AUC value improves when we correct the class distribution bias by doing re-sampling. The only exception is the kNN method that obtains similar AUC results in both cases. It might be due to the fact that kNN only uses the k nearest instances to classify a test instance. Hence, it could be less affected if the instances added by $SMOTE$ are not close to the test instances. Also, the values for the kappa statistics are slightly different if $SMOTE$ is applied. In any case, the results show that there is still space for improvement.

Let's go back to the confusion matrix included after the first experiment. It can be seen that the 80% of the TRSs whose true class is $MAYBE$ have been wrongly classified as YES. Note that if a TRS is

[4]There are two ways of extending the AUC for more than two classes: *each class against the rest* and *each class against each other*.

Technique	AUC	Kappa
DT	0.5520	0.2494
$DT + SMOTE$	0.6237	0.2804
SVM	0.4992	0.1681
$SVM + SMOTE$	0.5521	0.2471
NN	0.4958	0.2549
$NN + SMOTE$	0.6016	0.2176
kNN	0.6199	0.2919
$kNN + SMOTE$	0.6146	0.2602

Table 4: Performance of the four classification techniques with and without applying the oversampling method $SMOTE$ to the training dataset.

of class $MAYBE$, this only means that the termination tools have not been able to proof its termination. Therefore, the TRS can be, in fact, terminating or not. Based on this rational, we propose another learning scenario (see Figure 2).

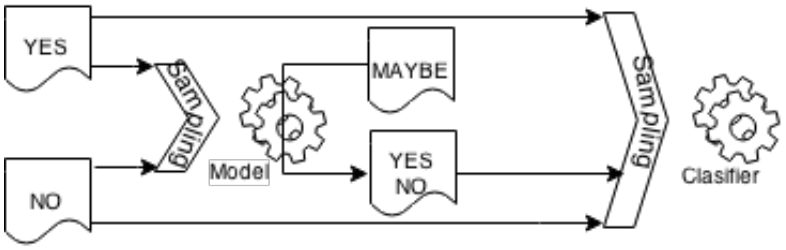

Figure 2: In a first step, a model using the instances of the classes YES and NO is learnt, by applying first the $SMOTE$ method. Then the model is used to re-label the instances $MAYBE$ as YES or NO. Finally, we construct a new training set by joining the new label instances with the original YES and NO instances, that is used to learn a binary classifier.

As we have mentioned in the introduction, the idea is to consider that the termination of TRS is a binary classification problem with only two classes (YES and NO) instead of a multi-class problem. So, we first separate the instances labeled as $MAYBE$ from the rest. It is done by splitting D into two disjoint sets: D_M that contains all the instances of D whose class is $MAYBE$, and ND, the subset with the rest of instances. Now, we remove the class labels of the instances in D_M. Then we proceed as in the previous experiments: we randomly split ND into a training set ND_{Train} (70%) which is used to learn a first binary classifier, and a test set ND_{Test} (30%) which will be used for the evaluation. Next, we apply the $SMOTE$ method to ND_{Train} and we train a binary classifier C. The instances in D_M are re-labeled by C and joined to ND_{Train} giving a new set $\overline{ND_{Train}}$ which is newly balanced with $SMOTE$ and used to train a new binary classifier NC. We evaluate NC by using ND_{Test}. The results are shown in Table 5.

We observe that the AUC results obtained by the $kNN + SMOTE$ model notably exceeds the results of the other methods. Also, the value of the kappa statistics is the best result not only in this experiment but also in the entire experimental study carried out. Nonetheless, in order to reduce the variability of the results and confirm the relative good outcomes exhibited in Table 5, more repetitions of the experiments should be carried out using different test and training sets.

Technique	AUC	Kappa
$DT + SMOTE$	0.6952	0.3800
$SVM + SMOTE$	0.6358	0.3352
$NN + SMOTE$	0.6767	0.3491
$kNN + SMOTE$	0.8722	0.7245

Table 5: Results of the binary classifier learnt after re-labelling the original instances of class *MAYBE* using the predictions of a binary classifier learnt with the original instances of the classes YES and NO. Before learn both classifiers, the training dataset is balanced applying the $SMOTE$ method.

4 Conclusion and Future work

This work represents the first attempt of using machine learning techniques for analysing the termination of a TRS. After the experimental analysis performed, we now have a better understanding of this problem: it is a highly imbalanced problem and the attention has to be put in those SRTs whose termination proofs finish with a successful demonstration.

On the other hand, the results obtained in the last experiment are encouraging. The achievements acomplished as the experiments progressed suggest that we are working in the right direction. Nevertheless, a lot of aspects remain unexplored. For instance, although in this paper we have used a relative small set of properties, some of them could be dependent or even redundant (for instance, linear, left-linear and right-linear). It could be interesting to study attribute dependencies or even to apply attribute selection algorithms that select the most significant attributes for the classification problem at hand. Also, we have used a very naive characterization of the recursive property. So, it should be further investigated the types of recursion.

Alternatively, we also plan to study the use of soft classifiers, which calculate for unseen instances the probability of belonging to a class instead of a class label. This kind of classifiers can be used to produce rankings.

Another possible source of data we want to investigate is the proof traces. The idea is to extract information about what techniques have been applied in a proof and which one has allowed to prove or disprove the termination of the TRS. The final goal is to use the discovered knowledge for making recommendations to the termination tools.

Finally, although in this paper we focus on termination of TRS, data mining techniques might be applicable to study the termination of other kind of programs (using the data in other categories of the termination competition).

Acknowledgements

We thank Raúl Gutiérrez for their insightful comments and very useful suggestions. This work was supported by the Spanish MINECO under grant TIN 2013-45732-C4-1-P, by Generalitat Valenciana PROMETEOII2015/013 and the REFRAME project, granted by the European Coordinated Research on Long-term Challenges in Information and Communication Sciences & Technologies ERA-Net (CHIST-ERA), and funded by the Ministerio de Economía y Competitividad in Spain (PCIN-2013-037).

References

[1] B. Alarcón, R. Gutiérrez, S. Lucas & R. Navarro-Marset (2011): *Proving termination properties with MU-TERM*. In: *Algebraic Methodology and Software Technology*, Springer, pp. 201–208.

[2] E. Albert, P. Arenas, M. Codish, S. Genaim, G. Puebla & D. Zanardini (2008): *Termination analysis of Java bytecode*. In: *Formal Methods for Open Object-Based Distributed Systems*, Springer, pp. 2–18.

[3] T. Arts & J. Giesl (2000): *Termination of term rewriting using dependency pairs*. *Theoretical Computer Science* 236(1), pp. 133–178.

[4] F. Baader & T. Nipkow: *Term Rewriting and All That, 1998*.

[5] N. Chawla, K. Bowyer, L. Hall & W. Kegelmeyer (2002): *SMOTE: synthetic minority over-sampling technique*. *Journal of artificial intelligence research*, pp. 321–357.

[6] Michael Codish, Jürgen Giesl, Peter Schneider-Kamp & René Thiemann (2012): *SAT solving for termination proofs with recursive path orders and dependency pairs*. *Journal of Automated Reasoning* 49(1), pp. 53–93.

[7] Jörg Endrullis, Johannes Waldmann & Hans Zantema (2008): *Matrix interpretations for proving termination of term rewriting*. *Journal of Automated Reasoning* 40(2-3), pp. 195–220.

[8] T. Fawcett (2003): *ROC graphs: Notes and practical considerations for researchers*. Technical Report, Technical report, HP Labs Tech Report HPL-2003-4.

[9] U. Fayyad, G. Piatetsky-Shapiro & P. Smyth (1996): *From data mining to knowledge discovery in databases*. *AI magazine* 17(3), p. 37.

[10] C. Ferri, J. Hernández-Orallo & R. Modroiu (2009): *An experimental comparison of performance measures for classification*. *Pattern Recognition Letters* 30(1), pp. 27 – 38.

[11] P. A. Flach, H. Blockeel, C. Ferri, J. Hernandez-Orallo & J. Struyf (2003): *Decision support for data mining: introduction to ROC analysis and its application*. Kluwer Academic Publishers.

[12] V. Ganganwar (2012): *An overview of classification algorithms for imbalanced datasets*. *International Journal of Emerging Technology and Advanced Engineering* 2(4), pp. 42–47.

[13] J. Giesl, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski & R. Thiemann (2014): *Proving termination of programs automatically with AProVE*. In: *Automated Reasoning*, Springer, pp. 184–191.

[14] J. Giesl, R. Thiemann, P. Schneider-Kamp & S. Falke (2004): *Automated termination proofs with AProVE*. In: *Rewriting Techniques and Applications*, Springer, pp. 210–220.

[15] H. Haibo & E.A. García (2009): *Learning from imbalanced data*. *Knowledge and Data Engineering, IEEE Transactions on* 21(9), pp. 1263–1284.

[16] J. Hernández, M.J. Ramírez & C. Ferri (2004): *Introducción a la Minería de Datos*. Pearson Prentice Hall.

[17] K. Hornik, C. Buchta & A. Zeileis (2009): *Open-source machine learning: R meets Weka*. *Computational Statistics* 24(2), pp. 225–232.

[18] M. Kuhn (2008): *Building predictive models in R using the caret package*. *Journal of Statistical Software* 28(5), pp. 1–26.

[19] V. López, A. Fernández, S. García, V. Palade & F. Herrera (2013): *An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics*. *Information Sciences* 250, pp. 113–141.

[20] S. Lucas (2005): *Polynomials over the reals in proofs of termination: from theory to practice*. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications* 39(3), pp. 547–586.

[21] Z. Manna & S. Ness (1970): *On the termination of Markov algorithms*. In: *Proceedings of the Third Hawaii International Conference on System Science*, pp. 789–792.

[22] C. Marché & H. Zantema (2007): *The termination competition*. In: *Term Rewriting and Applications*, Springer, pp. 303–313.

[23] A. Turing (1936): *On computable numbers, with an application to the Entscheidungsproblem*. J. of Math 58(345-363), p. 5.