SAT Feature Analysis for Machine Learning Classification Tasks

Marco Dalla[1,3], Benjamin Provan-Bessell[2], Andrea Visentin[2,3] and Barry O'Sullivan [1,2,3]

[1]SFI Centre for Research Training in AI, University College Cork, Ireland
[2]Insight SFI Research Centre for Data Analytics, University College Cork, Ireland
[3]School of Computer Science & IT, University College Cork, Ireland
{m.dalla,b.provanbessel,b.osullivan}@cs.ucc.ie, a.visentin@ucc.ie

Abstract

The extraction of meaningful features from CNF instances is crucial to applying machine learning to SAT solving, enabling algorithm selection and configuration for solver portfolios and satisfiability classification. While many approaches have been proposed for feature extraction, their relevance to these tasks is unclear. Their applicability and comparison of the information extracted and the computational effort needed are complicated by the lack of working or updated implementations, negatively affecting reproducibility. In this paper, we analyse the performance of five sets of features presented in the literature on SAT/UNSAT and problem category classification over a dataset of 3000 instances across ten problem classes distributed equally between SAT and UNSAT. To increase reproducibility and encourage research in this area, we released a Python library containing an updated and clear implementation of structural, graph-based, statistical and probing features presented in the literature for SAT CNF instances; and we define a clear pipeline to compare feature sets in a given learning task robustly. We analysed which of the computed features are relevant for the specific task and the tradeoff they provide between accuracy and computational effort. The results of the analysis provide insights into which features mostly affect an instance's satisfiability and which can be used to identify the problem's type. These insights can be used to develop more effective solver portfolios and satisfiability classification algorithms.

Introduction

Feature extraction from SAT instances is an essential task. Features are used to train machine learning models for a variety of tasks such as algorithm selection, classification of whether an instance is satisfiable or not, and to which class an instance is drawn from. Most existing approaches work on instances encoded in conjunctive normal form (CNF). While automatic feature extraction is an ongoing task (Dalla, Visentin, and O'Sullivan 2021), manual implementation and extraction of these features is still an effective way of judging the class of a problem. SATzilla (Xu et al. 2008) provides the benchmark for portfolio classification of SAT problems, providing a wide range of statistical, structural and probing features of a CNF that help describe the complexity and structure of the instance. However, other features have been more recently crafted (Alfonso and Manthey 2014) and (Ansótegui et al. 2017). Feature sets across studies are not easily comparable. Furthermore, their implementations are often outdated, possibly not in a executable state in their current form, and challenging to understand and use.

This paper presents the study of relevant features applied to two tasks of classification using machine learning algorithms and analyses their computational times. To perform this we wrote an easy-to-use Python library, called SATfeatPy [1], which implements the previously mentioned sets of features. A high-level description of the extracted features is provided, with a more in-depth explanation in the documentation. This work contributes to the ongoing effort of making SAT research and prototyping easier by offering functionalities in Python, e.g. PySAT (Ignatiev, Morgado, and Marques-Silva 2018).

We perform a study on five sets of features extracted with the tool from a dataset consisting of 3000 SAT and UNSAT CNFs, over ten different classes of problems. First, the feature sets are used to classify whether the problem is satisfiable (SAT) or not satisfiable (UNSAT) and to which problem class the instance belongs. The best approaches reach an accuracy of 99.5 % and 99.8%, respectively. Then, an investigation of the contribution of the relevant features to the two different classification tasks is performed. Finally, the computational time for each set of features is extracted and compared with the accuracy results for both classifications.

Related Works

We survey papers that devise hand-crafted features of SAT instances and utilise these features for various tasks such as algorithm selection and solver parameter optimisation. This review is split into two sections: the first discusses the theory and motivation of hand-crafted SAT instance features, and then applications of said features are analysed.

The first paper that shows the importance of extracting structural information from SAT instances is Mitchell et al. (Mitchell, Selman, and Levesque 1992). They underline a strong correlation between the hardness of a SAT instance and the ratio between the number of variables and clauses. Further work has since been done to estimate with increasing precision the empirical hardness of a SAT instance and,

[1]https://github.com/bprovanbessell/SATfeatPy

consequently, its likely solubility in meaningful time by a specific solver. In a later paper, Nudelman et al. (Nudelman et al. 2004), present 84 new statistical features. These stem from nine main categories: problem size, variable-clause-graph, variable graph, clause graph, balance, proximity to Horn formulae, LP-based, DPLL-probing, and local search probing. Further work by Ansotegui et al. (Ansótegui et al. 2017) and Alfonso et al. (Alfonso and Manthey 2014) introduces new features to describe the structure further and help classify CNF formulas. All the features presented so far are handcrafted, a more recent branch of research focuses on automated feature extraction using deep learning (Dalla, Visentin, and O'Sullivan 2021).

Extracting representative features of a boolean formula has many practical applications. One of the most relevant is portfolio algorithm selection (Leyton-Brown et al. 2003), a technique in which multiple algorithms (each of which specialises in solving/evaluating problem instances of a particular type) form a portfolio. The current problem instance is then classified, and the algorithm which performs the best on that class is then used on said problem instance. SATzilla-07 (Xu et al. 2008) is one of the first systems that leverages the use of these characteristics (features) to extract a measure of empirical hardness from satisfiability problems. This measure is then used to construct a per-instance algorithm portfolio that automatically selects the best SAT solver from a set of different SAT solvers, to minimise the solver's runtime on the instance.

Further work on automated algorithm selection, as well as other related approaches like algorithm configuration, planning and portfolio selection, is presented in the survey by Kerschke et al. (Kerschke et al. 2018). Their paper also presents an overview of the different features that can guide the selection of solvers for specific instances. Xu et al. (Xu, Hoos, and Leyton-Brown 2010) work combines the techniques of portfolio-based algorithm selection and automated algorithm configuration to produce a set of solvers capable of tackling instances in different problem domains. Lindauer et al. (Lindauer et al. 2015) leverage on a different set of features to perform an automated algorithm configuration of a popular portfolio-based solver. Finally, Misir et al. (Mısır and Sebag 2017) build a recommender system that selects the best solver through collaborative filtering. Approaches inspired by SATzilla, where they utilise different machine learning techniques to select the optimal algorithm, can be seen in the paper from Nikolic et al. (Nikolić, Marić, and Janičić 2013). An example of instance classification is demonstrated in a 2008 paper by Devlin et al.(Devlin and O'Sullivan 2008). They use multiple machine learning classifiers to directly predict SAT instance satisfiability by training them on the same features used by SATzilla. Furthermore, work that leverages deep learning techniques to perform algorithm selection, feature extraction and instance classification is presented in the papers of Loreggia et al. (Loreggia et al. 2016), and Bunz et al. (Bünz and Lamm 2017). Finally, a 2018 paper (Selsam et al. 2018) by Selsam et al. introduces Neurosat, a message parsing neural network that learns to provide a solution to SAT instances by only being trained to predict their satisfiability.

Feature Description

This section presents a brief description of the features extracted using SATfeatPy. We refer to the documentation and the original papers for a more detailed explanation of the features.

The features have been split into three main categories stemming from their original implementation. The first category is the feature set of SATzilla (Xu et al. 2008). SATzilla is a portfolio algorithm that extracts various features from preprocessed CNFs. This set includes features relative to the problem size, its variable clause (VCG) and variable graph (VG) implementation, balance features and its proximity to Horn formula and probing features that leverage on the application of the the Davis-Putman-Logeman-Loveland (DPLL) (Davis and Putnam 1960) algorithm and two local search algorithms.

The second category encompasses features from (Ansótegui et al. 2017). These include power law exponent, modularity, and fractal dimension, and are calculated using two new graphs, the Variable Incidence Graph (VIG) and the Clause Variable Incidence Graph (CVIG).

The third category of features has been implemented from (Alfonso and Manthey 2014). The authors introduce eight weighted graphs: the variable clause graph (VCG), variable (VG) graph, clause graph (CG), resolution graph (RG), binary implication graph (BIG), AND gate graph, BAND gate graph, and EXO gate graph. Statistics are calculated from the node degrees, and the edge weights (if present). These statistics include but are not limited to: minimum, maximum, mean, mode, median, standard deviation, number of zeros present, quartile values, rate of the mode value, and entropy. Additionally, a recursive weight heuristic is used to provide a score for each literal that represents the tendency for that literal to be present in a model of the formula.

Experimental Results

This section evaluates the performances of the techniques presented herein on two different tasks: SAT/UNSAT and instance category classification. Then a comparison of the computational effort required to extract them is performed. We considered five sets of features:

- *SATzilla base*: The base features from SATzilla (Xu et al. 2008) (not including the probing features). 38 features.
- *SATzilla full*: Base SATzilla features and probing features. 69 features.
- *ANT*: Features from Ansotegui et al. (Ansótegui et al. 2017). 4 features.
- *ALF*: Features from Alfonso et al. (Alfonso and Manthey 2014). 254 features.
- *All*: Features from all papers combined. 327 features.

We create a dataset of 3000 CNFs equally distributed over ten problem categories for the classification tasks. For each category we produced 150 SAT and 150 UNSAT instances. The dataset is generated using CNFgen (Lauria et al. 2017) and it is an extension of the one used in (Dalla, Visentin, and O'Sullivan 2021). CNFgen produces propositional formulas in the CNF DIMACS format that can be used as a benchmark

Feature set	RFC	XGB	Comp. Time (s)
SATzilla base	91.21%	92.34%	**0.03**
SATzilla full	**99.47%**	**99.54%**	3.18
ANT	86.14%	86.23%	0.15
ALF	92.04%	94.73%	119.54
All	99.11%	99.52%	122.89

Table 1: SAT/UNSAT classification accuracies and average computational times (in seconds) across feature sets.

for SAT solvers. It features several formula families (e.g. pigeonhole principle, ordering principle, k-coloring, etc.) as well as several formula transformations and the possibility of producing formulas directly from graph structures. Each set of features was used to train a Random Forest Classifier (RFC) and an XGBoost Classifier (XGB) (Chen and Guestrin 2016) to predict whether the instance was SAT or UNSAT and to which category it belonged. The most relevant features (computed as the features that have the highest impurity decrease, based on the mean and standard deviation of this decrease) within the Random Forest Classifier were recorded for each set of features for both classification tasks.

SAT/UNSAT Classification

This experiment evaluates if the feature extraction preserves the instance's satisfiability information. Table 1 shows the results of the experiment alongside the time required to extract the features. These results are the average accuracy based on ten runs of 10 cross-validation. The highest average accuracies are highlighted in bold.

SATzilla full performs the best, classifying 99.47% and 99.54% of the instances correctly with both the RFC and XGB, respectively. The performance difference with the complete set of features is minimal; we used the Bayesian statistical test presented in (Benavoli et al. 2017) to compare them statistically. Figure 1 shows the probability distribution of the comparison with a *region of practical equivalence* (rope) of $\pm 0.5\%$ accuracy. The result is that there is a 95% probability that the two techniques are equivalent from a practical perspective; for the XGBoost classifier, the percentage is over 99.9%. The difference between the best-performing sets and the rest is always statistically significant. This analysis shows that the probing features contained in *SATzilla full* are fundamental to assessing the satisfiability of the CNFs. The graph-based feature sets *ALF* and *ANT* preserve information relevant to this supervised task since they achieve high accuracy; however, their inclusion does not improve the results. The reasons could be that the extracted knowledge is already covered in *SATzilla full* or that the high number of redundant features leads to overfitting. Regarding the computational times, *SATzilla base* is considerably faster than all the other feature sets. The probing process slows *SATzilla full* considerably, and the complex graph creation makes *ALF* the most computationally expensive. Overall, this experiment shows that the features selected by Xu et al. (Xu et al. 2008) extract relevant information on the instance's satisfiability. The contribution of the other features does not seem to be relevant.

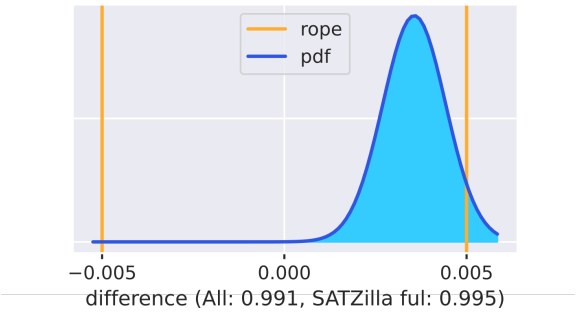

Figure 1: Bayesian comparison of *SATzilla full* and *All* on the SAT/UNSAT classification task

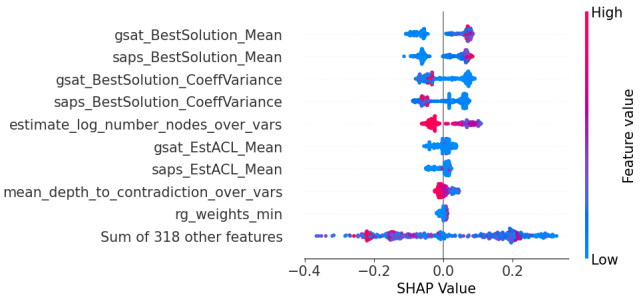

Figure 2: SHAP analysis performed for the Random Forest Classifier for the SAT/UNSAT classification task

We retrained the RFC over the *All* dataset with 100 estimators and performed a SHAP (SHapley Additive exPlanations) (Lundberg and Lee 2017) analysis to evaluate the features' importance. SHAP is a method used to explain the output of a machine learning model by calculating the contribution of each feature to the final prediction. It does this by applying game theory principles to assign credit to each feature based on its impact on the model's output. The result is a set of individual feature contributions that can be used to understand and interpret the model's behaviour.

Figure 2 shows the feature relevance in the decision process in order of importance. Each point represents a test instance; the colour stands for high/low values of that feature and the horizontal position if they contributed to classifying the CNF as SAT(right) or UNSAT(left). The eight most important features belong to the *SATzilla full* set. Six of them are probing features that are performed through a local search of the search space using the stochastic local search algorithms GSAT (Selman, Levesque, and Mitchell 1992) and SAPS (Hutter, Tompkins, and Hoos 2002). The algorithms are run multiple times until the search trajectory reaches a plateau that cannot be escaped within a given number of steps, and relevant statistics are extracted and averaged. The *estimate_log_number_nodes_over_vars* is a DPLL probing feature that approximates the size of the search tree over the number of variables; surprisingly, a bigger search tree contributes to a UNSAT classification of the instance.

Feature set	RFC	XGB
SATzilla base	99.64%	99.63%
SATzilla full	99.66%	99.67%
ANT	93.04%	93.11%
ALF	99.77%	99.66%
All	**99.81%**	**99.79%**

Table 2: Problem category classification accuracies across feature sets for both Random Forest and XGBoost classifiers.

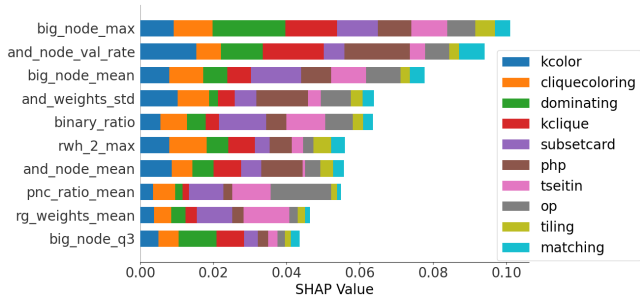

Figure 3: SHAP analysis performed for the RFC for the category classification task

Problem Category Classification

Understanding the instance's type is fundamental in algorithm portfolio approaches. Table 2 shows the results for this multi-class classification task.

The *All* set performed the best with both classifiers, with 99.81% and 99.79% accuracy for both RFC and XGB, respectively. This suggests that the feature sets help classify different types of instances correctly, contributing to overall better results. The task has more target labels compared to the previous one; however, all the approaches reach higher accuracy. *SATzilla base*, *SATzilla full*, and *ALF* perform excellently, showing that statistical features represent the problem structure more than its satisfiability. All the approaches except *ANT* are practically equivalent according to the Bayesian comparison.

Figure 3 shows the most relevant features according to a RFC trained on all the features; the different colours represent the feature's importance in the identification of that specific class. Interestingly, the majority of the features belong to graphs extracted in the *ALF* set. In particular, the binary implication graph (BIG), which contains all literals as vertices edges in between literals appearing in the binary clauses, and the AND-gate graph, which represents the relation between literals that belong to an AND-gate (l_0 iff $l_1 \dots l_k$) encoded in the CNF formula. There is a high variability in the feature importance depending on which class needs to be identified; this is the reason behind the better accuracy of the *All* set.

Feature Computation Time

This experiment evaluates the scalability of the techniques. We record the computational time to generate the features over a dataset of randomly-generated CNFs. The dataset comprises 300 instances with a consistent clauses-to-variables ratio of 4.2 and uniformly increased size (where size is the number of variables). The time to compute each feature set is plotted in Figure 4.

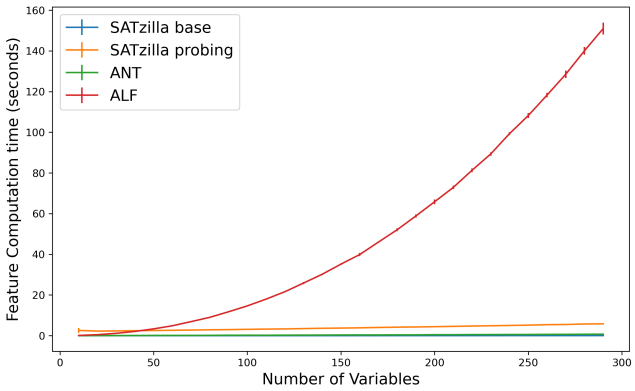

Figure 4: Time to generate features vs. size. The mean time to solve each set of instances is plotted in a line, and the variance is shown in the error bars.

The effort required to compute *ALF* features increases polynomially with the problem size, making it unsuitable for big instances. This is due to the many graphs that have to be created and the features number. The other approaches increase linearly with the instance size, with *SATzilla full* that has a minimum computational time represented by the probing timeout. *SATzilla base* is considerably faster than the other approaches.

Conclusions and Future Work

We offered a study of the performance for five sets of features presented in the literature on SAT/UNSAT and problem category classification. The features were extracted using an easy-to-use Python library, SATfeatPy. An in-depth statistical analysis carried out on a dataset of 3000 instances across ten problem classes and equally distributed between SAT and UNSAT showed high classification accuracy. We further investigated the features' relevance for two different tasks using explainability tools for machine learning. In addition, a time analysis showed the time needed to compute each feature set as the problem grew.

Our pipeline can be used to select feature sets for a given task and balance the trade-off between their accuracy and the computational effort. SATfeatPy can be used for fast prototyping and ideas validation of portfolio solutions. The readable and updated code and the documentation clarify existing inconsistencies.

Future works could focus on introducing and comparing additional sets of features and applying them to diverse tasks, i.e. solver selection and runtime prediction. Moreover, further trade-off analyses could focus on selectingi and computing only the relevant features for the pertinent job. Finally, applying new preprocessing approaches that can be combined with feature extraction techniques may yield more performing and accurate results.

Acknowledgments

This publication has emanated from research conducted with financial support of Science Foundation Ireland under Grant 16/RC/3918, 12/RC/2289-P2, and 18/CRT/6223, which are co-funded under the European Regional Development Fund.

References

Alfonso, E. M.; and Manthey, N. 2014. New CNF Features and Formula Classification. In *POS@ SAT*, 57–71.

Ansótegui, C.; Bonet, M. L.; Giráldez-Cru, J.; and Levy, J. 2017. Structure features for SAT instances classification. *Journal of Applied Logic*, 23: 27–39.

Benavoli, A.; Corani, G.; Demšar, J.; and Zaffalon, M. 2017. Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. *The Journal of Machine Learning Research*, 18(1): 2653–2688.

Bünz, B.; and Lamm, M. 2017. Graph Neural Networks and Boolean Satisfiability. *CoRR*, abs/1702.03592.

Chen, T.; and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Dalla, M.; Visentin, A.; and O'Sullivan, B. 2021. Automated SAT Problem Feature Extraction using Convolutional Autoencoders. In *33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021*, 232–239. IEEE.

Davis, M.; and Putnam, H. 1960. A Computing Procedure for Quantification Theory. *J. ACM*, 7(3): 201–215.

Devlin, D.; and O'Sullivan, B. 2008. Satisfiability as a Classification Problem. *Proc. of the 19th Irish Conf. on Artificial Intelligence and Cognitive Science*.

Hutter, F.; Tompkins, D. A. D.; and Hoos, H. H. 2002. Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT. In Van Hentenryck, P., ed., *Principles and Practice of Constraint Programming - CP 2002*, 233–248. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-46135-7.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python toolkit for prototyping with SAT oracles. In *International Conference on Theory and Applications of Satisfiability Testing*, 428–437. Springer.

Kerschke, P.; Hoos, H. H.; Neumann, F.; and Trautmann, H. 2018. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27: 3–45.

Lauria, M.; Elffers, J.; Nordström, J.; and Vinyals, M. 2017. CNFgen: A generator of crafted benchmarks. In *Proceedings of SAT*, 464–473.

Leyton-Brown, K.; Nudelman, E.; Andrew, G.; McFadden, J.; Shoham, Y.; et al. 2003. A portfolio approach to algorithm selection. In *IJCAI*, volume 3, 1542–1543.

Lindauer, M.; Hoos, H. H.; Hutter, F.; and Schaub, T. 2015. AutoFolio: An Automatically Configured Algorithm Selector. *J. Artif. Intell. Res.*, 53: 745–778.

Loreggia, A.; Malitsky, Y.; Samulowitz, H.; and Saraswat, V. A. 2016. Deep Learning for Algorithm Portfolios. In *Proceedings of AAAI*, 1280–1286.

Lundberg, S. M.; and Lee, S.-I. 2017. A Unified Approach to Interpreting Model Predictions. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Mısır, M.; and Sebag, M. 2017. Alors: An algorithm recommender system. *Artificial Intelligence*, 244: 291–314.

Mitchell, D. G.; Selman, B.; and Levesque, H. J. 1992. Hard and Easy Distributions of SAT Problems. In *Proc. of AAAI*, 459–465.

Nikolić, M.; Marić, F.; and Janičić, P. 2013. Simple algorithm portfolio for SAT. *Artificial Intelligence Review*, 40(4): 457–465.

Nudelman, E.; Leyton-Brown, K.; Hoos, H. H.; Devkar, A.; and Shoham, Y. 2004. Understanding Random SAT: Beyond the Clauses-to-Variables Ratio. In Wallace, M., ed., *Proceedings of CP*, 438–452.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, 440–446. AAAI Press. ISBN 0262510634.

Selsam, D.; Lamm, M.; Bünz, B.; Liang, P.; de Moura, L.; and Dill, D. L. 2018. Learning a SAT Solver from Single-Bit Supervision. *CoRR*, abs/1802.03685.

Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2010. Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In Fox, M.; and Poole, D., eds., *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: portfolio-based algorithm selection for SAT. *Journal of artificial intelligence research*, 32: 565–606.