Proceedings of the 11th International Conference on Applied Informatics
Eger, Hungary, January 29–31, 2020, published at http://ceur-ws.org

Classification of SAT Problem Instances by Machine Learning Methods

Márk Danisovszky, Zijian Győző Yang, Gábor Kusper

Eszterházy Károly University, Faculty of Informatics, Hungary
`danisovszky.mark`,`yang.zijian.gyozo`,`kusper.gabor`@uni-eszterhazy.hu

Abstract

Efficient SAT solving is critical in many practical applications. State-of-the-art SAT solvers can solve SAT problems with more than 100.000 variables and millions of clauses. They have usually more than 10 options, and we might combine them. We need those options to get better runtime. Our experience shows that we can get significant speed up if we use the suitable options. Our goal is to create a neural network based system, which can select the best option configuration for the input SAT instance. As a first step, we created a system that can predict the type of a SAT problem. Our system has two main tools: CNFStats and the prediction tool. The CNFStats can compute several properties of the input SAT instance. For example, how many variables, clauses, unit, binary clause are in the input. How many horn clauses, so called black and white clauses are in the input. These properties generated by CNFStats are the input for the prediction tool. The prediction tool is a multi-layer perceptron neural network, which is able to do classification over SAT problem instances found on the SATLIB webpage, see: `https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`. The neural network is written in .NET from scratch. It is easy to configure by using a webpage. The user can give the number of layers, number of neurons, and the activation function, train it, test it and save it. In the prediction tool we use those 48 properties which is computed by the CNFStats tool. We used the half of the input problems from SATLIB to train the neural network. Classes were the same as in the webpage. The most successful neural network was using 2 hidden layers, 125 neurons in each hidden layer, the activation function was the sigmoid function (other activation functions did not perform well in these settings), 10.000 training runs with 0.033 learning rate. The accuracy was 95%. The second best neural network was the same but with 1 hidden layer and 250 neurons.

Keywords: SAT solvers, neural network, classification

1. Introduction

Efficient SAT solving is critical in many practical applications. State-of-the-art SAT solvers can solve SAT problems with more than 100.000 variables and millions of clauses. They have usually more than 10 options, and we might combine them. We need those options to get better runtime. Our experience shows that we can get significant speed up if we use the suitable options. Our goal is to create a neural network based system, which can select the best option configuration for the input SAT instance. As a first step, we created a system that can predict the type of a SAT problem.

In our research we have created features for machine learning algorithms, then, using these features training corpora were built. We have tried more machine learning methods. At last, but not least, we have evaluated our experiments.

2. Related Work

It is well known that different SAT solvers have many different options which corresponds to different solution strategies. Each of them is optimised for some SAT problem classes. So choosing the right options give better runtime results. This idea was used at The Configurable SAT Solver Challenge (CSSC), see [4]. They run SAT solvers after a fully automated configuration step.

This is called algorithm configuration and this technique seem to be effective in case of several SAT solvers. For example, Hutter et al. [3] configured the algorithm Spear on formal verification instances, achieving a 500-fold speedup. Algorithm configuration has also enabled the development of general frameworks for stochastic local search SAT solvers that can be automatically instantiated to yield state-of-the-art performance on new types of instances, like SATenstein [5]. Algorithm configuration can be seen as a special case of automatic data correction, see for example [10], where a neural network is used to correct sensor data automatically.

On the other hand neural networks was trained to recognize interesting properties of SAT problems. For example Evans et al. constructed a neural network to recognize logical entailment[1]. NeuroSAT is designed to guess the unsatisfiable core of SAT instances [8, 9].

In the research of Evans et al. [12], a new process was performed to recognising logical entailment. LSTM and Convolution network were used to capture and exploit heterogeneous and deeply structured syntax of logic.

In our experiments, we used the neural network as traditional machine learning method to train human created attributes that extracted from logical expressions.

3. CNFStats and corpora

Our prediction system is based on machine learning. Our classification system has two main tools: CNFStats, see `http://fmv.ektf.hu/files/CnfStats.java`,

Name	Description
number Of Variables	Number of variables.
number Of Clauses	Number of clauses.
number Of K Clauses (1-13)	How many 1, 2 ... 12. literal clauses. The 13 or more literal clauses are count as 13 literal clauses.
number Of Black Clauses	Number of Black clauses.
number Of White Clauses	Number of White Clauses.
number Of Definite Horn Clauses	Number of Definite Horn Clauses.
number Of Strait Clauses	Number of Strait Clauses.
number Of Positive Literals	number of Positive Literals.
number Of Negative Literals	Number of Negative Literals.
ratio Of Clauses And Variables	Ratio of clauses and variables.
ratio Of K Clauses (1-13)	Ratio of K Clauses (K=1-13).
ratio Of Black Clauses	Ratio of Black Clauses.
ratio Of White Clauses	Ratio of White Clauses.
ratio Of Definite Horn Clauses	Ratio of Definite Horn Clauses.
ratio Of Strait Clauses	Ratio of Strait Clauses.
ratio Of Positive Literals	Ratio of Positive Literals.
ratio Of Negative Literals	Ratio of Negative Literals.
mayBe PigeonHole	Is it a PigeonHole problem?
mayBe Random3SAT	Is it a Random3SAT problem?
mayBe RandomAIM	Is it a RandomAIM problem?
mayBe NemesisFormula	Is it a NemesisFormula problem?
mayBe Dubois	Is it a Dubois problem?

Table 1: Features

and prediction tool, see `http://fmv.ektf.hu/files/SAT-CLASSIFIER.rar`. The CNFStats extracts features from the input SAT instance. For example, how many variables, clauses, unit, binary clause are in the input. We extract 48 features, the complete feature list can be found in Table 1. These extracted features are the inputs of a machine learning algorithm. The output is the result of the classification, so it is a type of SAT problems.

First, our CNFStats tool reads a DIMACS (standard SAT format) file. The DIMACS file contain a propositional logic formula in conjunctive normal form (CNF). The DIMACS file format represents clauses as a line, which is a list of integers terminated by a zero. Positive integers represent positive literals, negative ones represent negative literals. The number of variables is usually between 10 and 100,000 and the number of clauses is usually between 100 and or even few millions, so we cannot use a DIMACS file directly to train a neural network. Therefore, we need to create features from these variables and clauses. Our aim was to create relevant features that can help a classifier in its work. In Table 1 we can see all the 48 features that CNFStats extracts.

The last 5 features in Table 1 are invented by an expert, Gábor Kusper, who is one of the authors, by closely investigating the corresponding DIMCAS files. For example the 'mayBe PigeonHole' feature is computed by a 12 lines Java method which uses the other features. It tries to guess whether the input SAT problem is a Pigeonhole SAT problem or not, see `https://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/DIMACS/PHOLE/descr.html`.

This Java code is a segment of the CNFStats tool. It is the source code of the method which computes the 'mayBe PigeonHole' feature:

```
private boolean mayItBePigeonHole() {
  int binary = numberOfKClauses[2];
  if (binary != numberOfBlackClauses) return false;
  if (numberOfBlackClauses + numberOfWhiteClauses !=
      numberOfClauses) return false;
  int nonBinary = numberOfClauses - binary;
  if (nonBinary < 4) return false;
  int k = nonBinary-1;
  k = k > 12 ? 13 : k;
  if (numberOfKClauses[k] != k+1) return false;
  return true;
}
```

To invent this method an expert used to need 2 hours of work, and this was the simplest one. We invented these 'mayBe' features to check whether the 48 statistics what we compute are suitable to classify a SAT instance or not. Since we could do that for 5 classes by hand, it seems that the measured features are sufficient.

Using these 48 features, we created our training sets. We built 2 different corpora from the SAT problems from the SATLIB webpage, see `https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`.

1. **SAT33**: 3557 instances, 33 classes: 3CNF, 3SAT1K, 5SAT500, 7SAT90, AIM, AIS, BEJING, BF, BMC, BMS, BW, CBS, CF, DUBOIS, GCP, HANOI, II, JNH, K3, LABS, LOGISTICS, LRAN, PARITY, PHOLE, QG, QI-CRAFTED, QI-ISO, QUEENS, RND3SAT, SSA, SW-GCP, UNIF-K5, WND

2. **SAT19**: 1038 instances; 19 classes: AIM, AIS, BEJING, BF, BMC, BMS, BW, CBS, DUBOIS, GCP, HANOI, II, JNH, LOGISTICS, LRAN, PARITY, QG, RND3SAT, SW-GCP

We decided to have 2 different corpora because the learning time in case of SAT19 was significantly smaller than in case of SAT33. For example we decided not to include QI-CRAFTED and QI-ISO into SAT19 because we had 40GB CNF files for them. In case of 3SAT1K, 5SAT500 the reason was that they are very similar as we can see in Figure 4. The other reason is that classes in SAT33 are less widely known. It includes also CNF files from the SAT race competition, see http://www.satcompetition.org/. On the other hand, all classes from SAT19 can be found on the SATLIB page.

In Figure 1 we can see the number of instances for each class in our corpora. Since the distribution is not uniform, we had to use 10-fold cross-validation in our experiments.

Using the training sets, prediction models are trained and our prediction tool can predict a concrete type of the input SAT problem given as a DIMACS file.

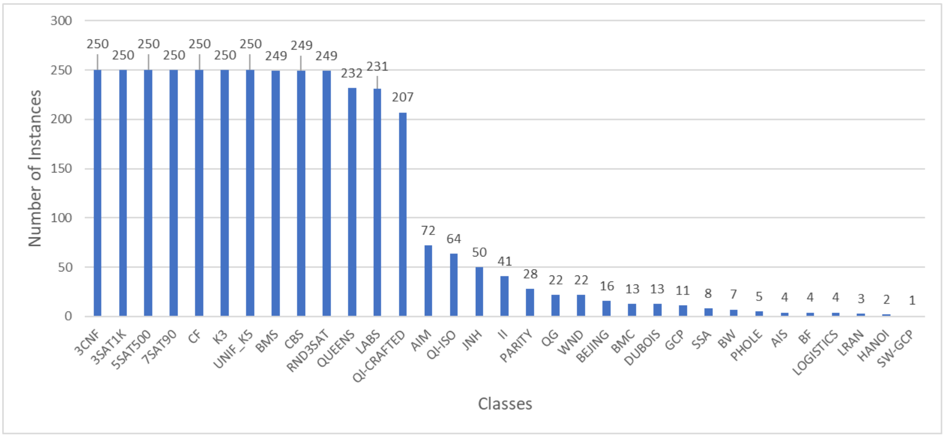


Figure 1: Number of instances for each SAT class

4. Reusable User Friendly MLP Neural Network Framework

For training a classifier neural network, we have implemented a reusable user friendly multilayer perceptron neural network framework in C# language[1]. Neural network is a commonly used method in research projects, but building a neural network is difficult and programming knowledge is needed. Thus, we have created an user friendly software for users who would like to build a neural network for our CNFStats without programming skill. But the software can be easily changed for an other kind of task.

In our software, first, the user can choose the output classes for the neural network, then we have to add the folder of CNF sources and the output folder for each classes. After the input settings, the second step is to build our neural network. The user can set the number of hidden layers, then set the number of neurons of the input layer, the hidden layers and the output layer. Activation function can be chosen as well. After building the architecture of the network,

[1]https://github.com/DanisovszkyMark/EKE-PROBLEM-CLASSIFIER-WITH-NEURAL-NETWORK_V2

the user can set the training options and start the training steps. At the end of training, the user can test the learned model.

5. Methods and Experiments

First, we did preprocess experiments to find the best hyperparameters for our neural network. For this task, we tried different settings and architectures of neural network:

- 1 hidden layer:
  - Number of neurons: 35, 150, 200, 250, 300, 500
  - Epochs: 1000 - 10000
- 2 hidden layer
  - Number of neurons: 125-125, 150-100, 200-50
- 3 hidden layer
  - Number of neurons: 100-100-50, 100-75-75, 125-75-50
- Corpus: Training set: 80%, Test set: 20%

As we can see in Figure 3, neural network with 1 hidden layer (the green bars) could gain same performance against the 2 (the blue ones) or 3 hidden layers (the yellow ones), and the training time of 1 hidden layer is also much faster although in all 3 scenarios we had 250 neurons in total. Thus in our further experiments, we used only one hidden layer in our neural network. Furthermore, in Figure 3 the green bars show that among the neural networks with 1 hidden layer the one with 250 neurons achieved the highest result. Figure 2 shows that 4000 epoch steps are enough for the training.

Using the pretrained neural network we have built our neural network classification models.

Furthermore, we have tried also different kinds of machine learning algorithms for this task. Beside the neural network, we have tried 4 basic machine learning classifiers [11]:

- Naive Bayes: probabilistic classifier based on Bayes' theorem with strong independence assumptions between the features.
- Support Vector Machine: supervised machine learning algorithm, it uses a non-probabilistic binary linear classifier. The separated categories are divided by a clear gap that is as wide as possible.
- Decision Tree (J48): an open source Java implementation of the C4.5 algorithm, which is used to generate a decision tree.

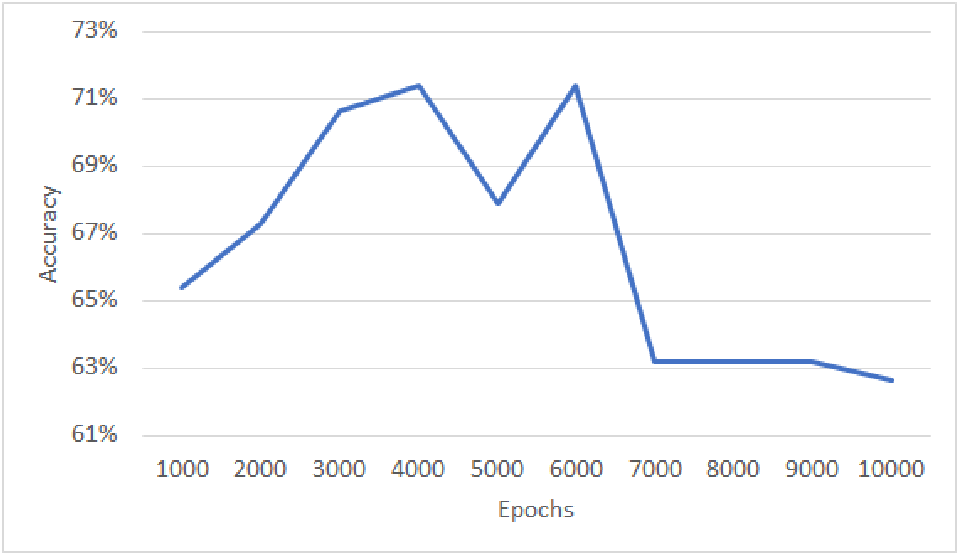


Figure 2: Performance of epochs

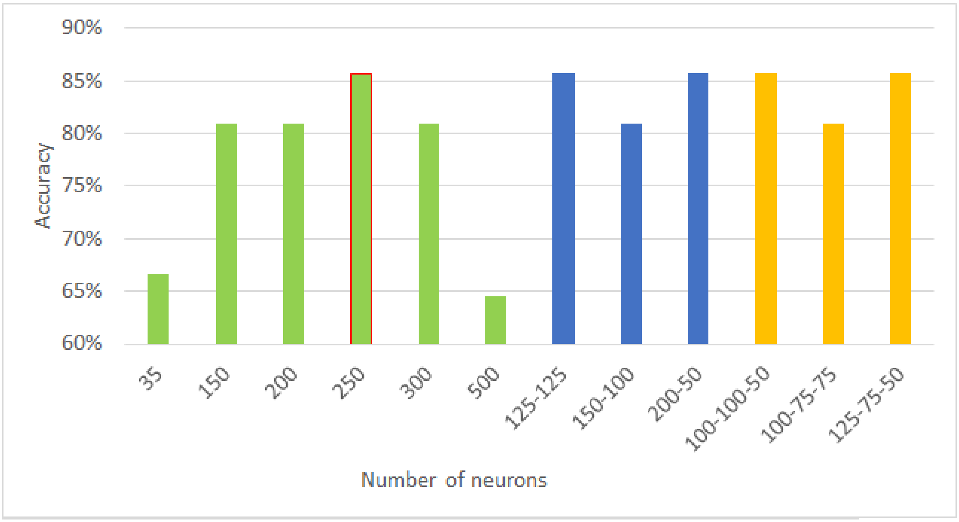


Figure 3: Experiment of hidden layer

- Random Forest: it constructs multitude of individual decision trees during training and at prediction phase, each individual tree in the random forest spits out a class prediction and the class with the most votes becomes the model's prediction.

Using these machine learning methods, we trained classifier models on SAT19 and SAT33. For training and testing, we used the Weka 3 open source machine learning software [2].

6. Results and Evaluation

For evaluation, we used the "Correctly Classified Instances" metric. In all cases we used 10-fold cross-validation. In Table 2 we can see the evaluation of the machine learning algorithms. In our experiment, the Random forest achieved the best result, close to 99% accuracy.

	CCI	
	SAT33	SAT19
MLP	85.71%	71.37%
SVM	80.49%	73.60%
Naive Bayes	90.95%	73.60%
J48	98.76%	98.84%
Random forest	**98.90%**	**99.03%**

Table 2: Performance of the machine learning methods

In Table 3 we can see the relevance of the features computed by the CnfStats tool. This table was created by the decision tree (J48) technique. The depth of the decision tree was 9, this table show the features used at each decision level. The first decision is made whether the "maybe random 3 sat" feature was 0 or 1. This is very important, because lots of classes are some variant of the random 3 SAT problem. We can see that the "number of kclauses" feature was used on many levels so this is an interesting feature.

We also did error analysis. In Figure 4 we can see the confusion matrix of the SAT33 (left) and the SAT19 (right) of the random forest experiments. In these matrices only the mistakes are shown. The columns are the predicted classes, the rows are the original classes. The most outstanding mistake is CBS and QI-CRAFTED. Our method predicted 12 times CBS, but it was originally K3 and 5 times QI-CRAFTED, but it was QI-ISO.

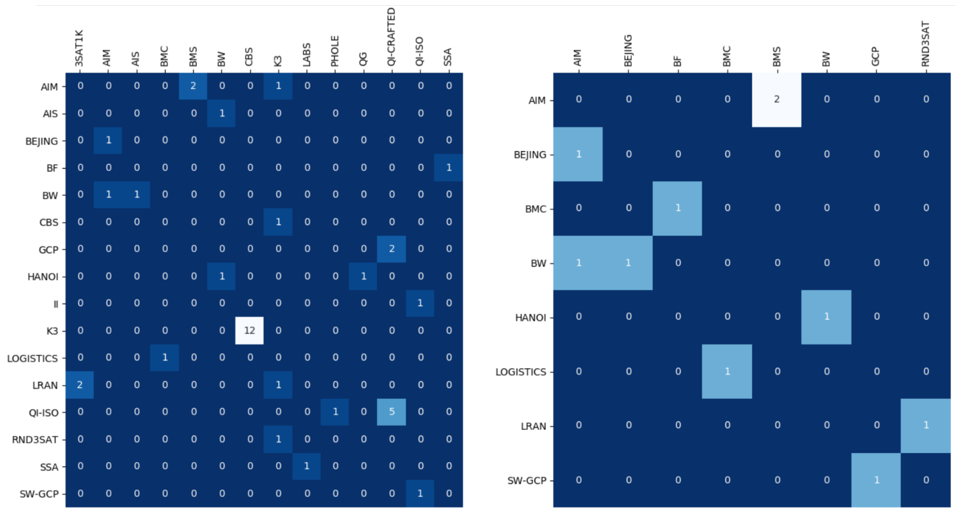
7. Conclusion

In our research we have created a SAT problem classifier. For this task we did experiments in feature engineering and feature selection. We have tried five different kinds of machine learning algorithm. Among the tried methods, the random forest gained the best result. We could achieve **98.9%** accuracy at SAT33 and **99.03%** accuracy at SAT19. Furthermore we have implemented a reusable user friendly MLP neural network framework for classifying SAT problems.

As a future work we would like to use also other techniques for algorithm configuration, like optimization modulo theories [6] using for example Puli [7].

	features
1	maybe random 3 sat
2	ratio of strait clauses, number of variable
3	number of kclauses 8 number of black clauses ratio of kclauses 3 number of clauses ratio of clauses and variable
4	ratio of kclauses 4 number of variables ratio of kclauses 6 number of kclauses 3 number of clause
5	ratio of kclauses 1 maybe dubois number of variables maybe nemesis formula number of kclauses 4 ratio of kclauses 6
6	number of kclauses 14 ratio of strait clauses number of variables number of kclauses
7	number of definite horn clauses number of kclauses 5 number of kclauses
8	number of white clauses ratio of kclauses 14 maybe nemesis formula
9	ratio of kclauses 14

Table 3: Features sorted by relevance



	3SAT1K	AIM	AIS	BMC	BMS	BW	CBS	K3	LABS	PHOLE	QG	QI-CRAFTED	QI-ISO	SSA
AIM	0	0	0	0	2	0	0	1	0	0	0	0	0	0
AIS	0	0	0	0	0	1	0	0	0	0	0	0	0	0
BEJING	0	1	0	0	0	0	0	0	0	0	0	0	0	0
BF	0	0	0	0	0	0	0	0	0	0	0	0	0	1
BW	0	1	1	0	0	0	0	0	0	0	0	0	0	0
CBS	0	0	0	0	0	0	0	1	0	0	0	0	0	0
GCP	0	0	0	0	0	0	0	0	0	0	0	2	0	0
HANOI	0	0	0	0	0	1	0	0	0	0	1	0	0	0
II	0	0	0	0	0	0	0	0	0	0	0	0	1	0
K3	0	0	0	0	0	0	12	0	0	0	0	0	0	0
LOGISTICS	0	0	0	1	0	0	0	0	0	0	0	0	0	0
LRAN	2	0	0	0	0	0	0	1	0	0	0	0	0	0
QI-ISO	0	0	0	0	0	0	0	0	0	1	0	5	0	0
RND3SAT	0	0	0	0	0	0	0	1	0	0	0	0	0	0
SSA	0	0	0	0	0	0	0	0	1	0	0	0	0	0
SW-GCP	0	0	0	0	0	0	0	0	0	0	0	0	1	0

	AIM	BEJING	BF	BMC	BMS	BW	GCP	RND3SAT
AIM	0	0	0	0	2	0	0	0
BEJING	1	0	0	0	0	0	0	0
BMC	0	0	1	0	0	0	0	0
BW	1	1	0	0	0	0	0	0
HANOI	0	0	0	0	0	1	0	0
LOGISTICS	0	0	0	1	0	0	0	0
LRAN	0	0	0	0	0	0	0	1
SW-GCP	0	0	0	0	0	0	1	0

Figure 4: Confusion matrix of the SAT33 (left) and the SAT19 (right)

References

[1] R. Evans, D. Saxton, D. Amos, P. Kohli, and E. Grefenstette, Can neural networks understand logical entailment? *In 6th International Conference on Learning Representations, ICLR 2018*, (2018), `http://arxiv.org/abs/1802.08535`.

[2] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten, The WEKA Data Mining Software: An Update, *SIGKDD Explor. Newsl., Volume 11:1, ISSN 1931-0145*, (2009), pp. 10–18.

[3] F. Hutter, D. Babic, H. Hoos, and A. Hu, Boosting verification by automatic tuning of decision procedures, *Formal Methods in Computer Aided Design (FMCAD'07)*, (2007), pp. 27-–34.

[4] F. Hutter, M. T. Lindauer, A. Balint, S. Bayless, H. H. Hoos, K. Leyton-Brown, The Configurable SAT Solver Challenge (CSSC), *CoRR*, abs / 1505.01221, (2015), `http://arxiv.org/abs/1505.01221`.

[5] A. KhudaBukhsh, L. Xu, H. Hoos, and K. Leyton-Brown, SATenstein: Automatically building local search SAT solvers from components. *Proceedings of IJCAI'09*, (2009), pp. 517-–524.

[6] G. Kovásznai, Cs. Biró, and B. Erdélyi, Generating Optimal Scheduling for Wireless Sensor Networks by Using Optimization Modulo Theories Solvers *CEUR WORKSHOP PROCEEDINGS 1889*, (2017), pp. 15–27.

[7] G. Kovásznai, Cs. Biró, and B. Erdélyi, Puli – A Problem-Specific OMT solver *Proceedings of SMT 2018*, (2018), Paper: 362.

[8] D. Selsam, M. Lamm, B. Bünz, P. Liang, Leonardo de Moura, and D. L. Dill. Learning a SAT Solver from Single-Bit Supervision, *ICLR 2019 Conference*, (2019), `https://openreview.net/forum?id=HJMC_iA5tm`.

[9] D. Selsam, N. Bjørner, Guiding High-Performance SAT Solvers with Unsat-Core Predictions, *SAT 2019: Theory and Applications of Satisfiability Testing – SAT 2019*, (2019), pp. 336–353.

[10] T. Tajti, G. Geda, T. Balla, and Gy. Vad, Indoor localization using NFC and mobile sensor data corrected using neural net *Proceedings of ICAI 2014*, vol. 1-2, (2014) pp. 163–169.

[11] I. H. Witten, E. Frank, and M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, *Morgan Kaufmann Publishers Inc., ISBN 0123748569*, (2011), book.

[12] Richard Evans and David Saxton and David Amos and Pushmeet Kohli and Edward Grefenstette, Can Neural Networks Understand Logical Entailment?, *International Conference on Learning Representations*, (2018), book.