Design Project Report

Roll Number	**B20CS076**
Name	**Suyash Bansal**
Title	**Development of Hardware Trojan detection Technique**

INTRODUCTION

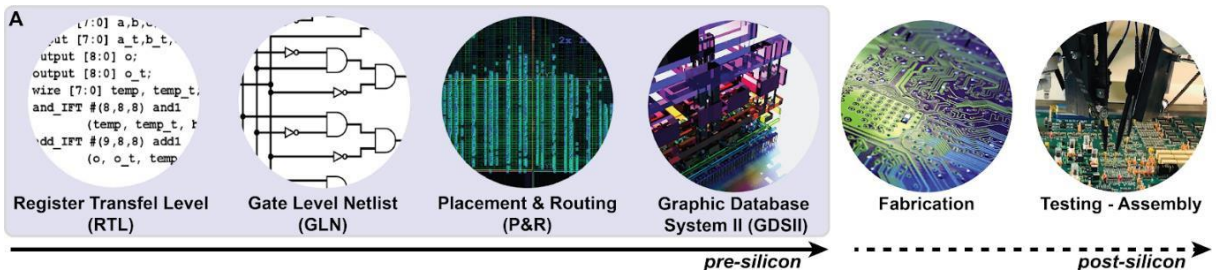
Motivation and preliminaries

Our century has been recognized as the century of technological evolution, and not unjustly. Every year the need for more technological ideas leads, to even more complex and sophisticated software and hardware. Also, it must be mentioned that a significant reason for this technological complexity, is the usage of tiny sensors to even more devices in a combination of Data Analysis (DA) and Artificial Intelligence (AI).

This evolution of technological complexity it was not possible not to affect the design industry. Specifically, design companies are not able to manufacture their own circuits. To overcome this problem, they outsource the design and more frequently the fabrication of their ICs to manufacturing companies, thus risking potential security issues. The newest example of security issues consists of the HTs.

Specifically, design companies are not able to manufacture their own circuits. In order to overcome this obstacle, they outsource the design and more frequently the fabrication of their ICs to manufacturing companies, thus risking potential security issues. Apart from this, Intellectual Property (IP) cores are obtained from various third-party suppliers so as to produce modern and complex ICs with the scope of making security issues even more sensitive. The newest example of security issues consists a new type of virus known as HT.

HTs are viruses which are associated with changes affecting the circuits during the design and/or fabrication phase and are usually introduced by an untrusted design foundry or design software. Due to the complex nature of modern circuits, HTs can be introduced into every IC development phase (Figure 1) and stay dormant until triggered by a wide range of activation mechanisms. HTs are associated with unexpected IC malfunctions, circuit destruction, as well as the leakage of sensitive information regardless of the encryption state [1]. In the recent years in electronics' field, HT infection became a serious issue with chances of being a severe threat from a technological but also social aspect.



IC development phase

Tools used

The code was arranged around Spyder. Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package. Furthermore, Spyder offers built-in integration with many popular scientific packages, including Pandas, Ipython, QtConsole, Matplotlib, SymPy, and more. In our work, we mostly make use of Scikit-learn which is a free ML library for Python. It features various algorithms SVM, RF, and KNN, and it also supports Python numerical and scientific libraries like NumPy and SciPy. We can use separate working areas for the Variables values, the command line and code editor.

Dataset Construction

Experiments were carried out on a dataset that was created by research benchmarks from Trust-Hub, a certified public library with Trojan free (TF) and Trojan Infected (TI) circuits. For the creation of our dataset we used all the circuits from Trust-Hub, approximately 1.000 circuits. Twenty-one (21) of them refers to TF circuits and all the other refers to TI circuits. Then, using industrial circuit design tools (DC compiler Synopsys) and our own scripts, we created the dataset for the design phase known as Gate Level Netlist (GLN). We consider dealing with characteristics about the area, power and time of each circuit. The feature extraction process created the final dataset, consisted of 50 characteristics.

Dataset Preprocessing and Feature Extraction

The dataset had some minor inconsistencies such as missing features that we removed during the dataset cleaning phase. It includes 50 features and after dropping the highly correlated ones - with at least 95% correlation, in order to make our algorithm more cost and time efficient - it consists of the following 24 features, plus the feature with the target values:

Area	Power (1)	Power (2)	Power (3)
Number of ports	Net Switching Power	Cell Internal Power	Black_Box Total Power
Number of sequential cells	Cell Leakage Power	Memory Switching Power	Clock_Network Internal Power
Number of macros/black boxes	IO_Pad Internal Power	Memory Leakage Power	Clock_Network Switching Power
Number of references	IO_Pad Switching Power	Memory Total Power	Clock_Network Leakage Power
Macro/Black Box area	IO_Pad Leakage Power	Black_Box Internal Power	Sequential Internal Power
	IO_Pad Total Power	Black_Box Switching Power	Sequential Leakage Power
	Memory Internal Power	Black_Box Leakage Power	

Features set

As we see above, there were only 21 out of 1.000 TF labeled instances, with a particular type of circuit. In order to balance the ratio between TF and infected, we used a reproduced technique in order to reproduce each TF multiple times, to match the total number of TI of the same circuit category. This type of technique is not the optional, but recommended under unbalanced datasets.

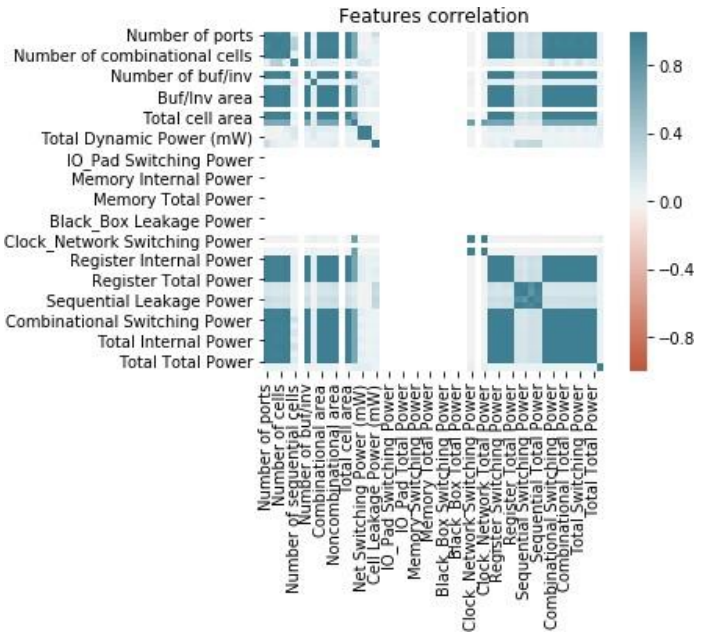
As for the features dropping step, when we examine the features of a dataset, some of them might not be useful to make the necessary prediction. Correlation refers to how close two variables (features for our problem) are to having a linear relationship with each other. Features with high correlation are more linearly dependent (e.g. $y = 3 \cdot x$) and hence have almost the same effect on the dependent variable, in contrast with two features that are non-linearly dependent (e.g. $y = x^2$). So, when two features have high correlation, we can drop one of the two features.

We use the LabelEncoder class provided by Scikit Learn library, in order to transform the nominal values (numeric codes used for labelling or identification, such as strings, datetimes, etc) to numeric and enable the algorithm to perform arithmetic operations on these values. Next, we create the correlation matrix, which is computed as follows:

Correlation Matrix = $[Cor(x_j, x_k)]_{ik}$, where x_j and x_k are the vectors of two features' values and

$$\mathrm{Cor}(x_j, x_k) = \sum_{i=1}^{n} (x_{ij} - mean(x_j)) * (x_{ik} - mean(x_{ik})) * \left(\sum_{i=1}^{n} (x_{ij} - mean(x_j))^2\right)^{-\frac{1}{2}} * \left(\sum_{i=1}^{n} (x_{ik} - mean(x_k))^2\right)^{-\frac{1}{2}}$$

and visualize it through the correlation heat-map, as we can see on the figure below.



Plot of features correlation

Next, we need to shuffle the data before each iteration. During the training, it is important to shuffle the data so that the model does not learn specific pattern of the dataset. If the model learns the details of the underlying pattern of the data, it will have difficulties to generalize the prediction for unseen data. This is called overfitting. The model performs well on the training data but cannot predict correctly for unseen data.

Lastly, we scaled our data using the MinMaxScaler class of Scikit Learn in the range of (0,1) for easier convergence. MinMaxScaler is a good choice if we want our data to have a normal distribution or want outliers to have reduced influence. Specifically, if max(X) and min(X) are the maximum and minimum values that appear in all dataset for a given feature X, a value X is replaced by newX using the equation:

$$newX = \frac{X - min\ (X)}{max(X) - minx(X)}$$

This procedure was not implemented for the SVM methodology, as SVMs are scale invariant.

Algorithms used

Based on the previous research papers, we decided on the use of eight algorithms to focus on the issue and we will analyze below the work implemented in each one:

MLP
RF
SVM
GB
KNN
LR
XGB

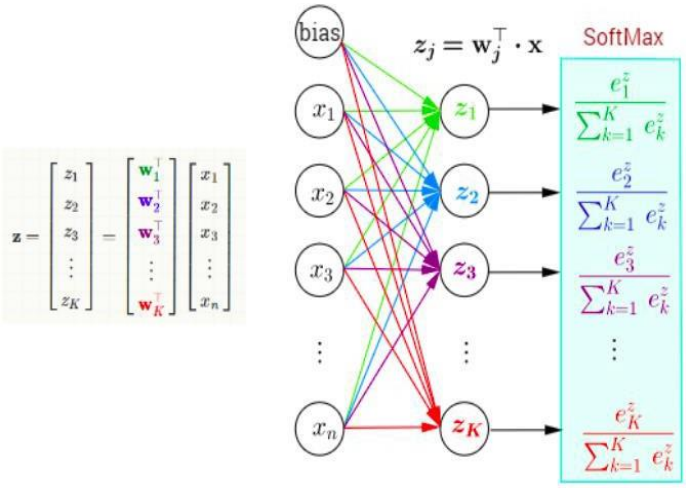
ML algorithms used

MLP

In our case, we construct a probability distribution of the 2 possible values (0 for Trojan-free, 1 for Trojan-infected) by running the outputs through a softmax activation function.

Simply speaking, the sigmoid function only handles two classes, which is not what we expect. The softmax function squashes the outputs of each unit to be between 0 and 1, just like a

sigmoid function. But it also divides each output such that the total sum of the outputs is equal to 1.



Softmax activation function

The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true. Mathematically the softmax function is shown below, where z is a vector of the inputs to the output layer (if you have 10 output units, then there are 10 elements in z). And again, j indexes the output units, so j = 1, 2, ..., K

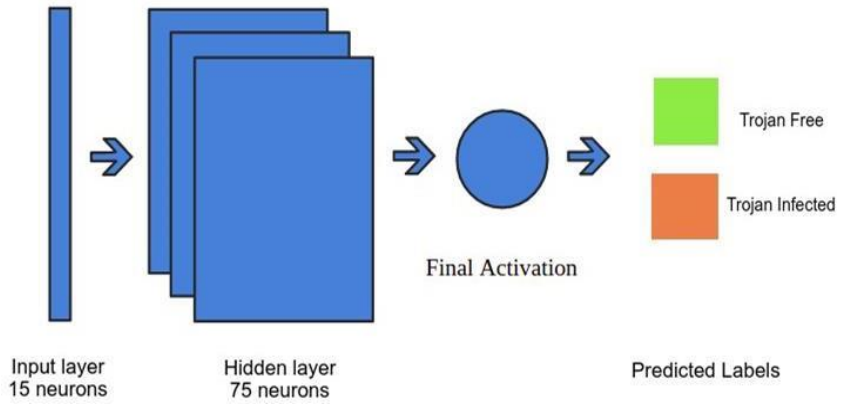
$$\sigma(j) = \frac{e^{w_j^T * x}}{\sum_{k=1}^{K} e^{w_j^T * x}} = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

The model was compiled with Adam optimizer, since it is appropriate for problems with very noisy/or sparse gradients, and a categorical cross-entropy loss function, since we have multiple classes, and trained in batches of 150 instances for 50 iterations. For the fully connected neural networks architecture we used:

- A Dense input layer of 15 neurons and rectifier linear unit activation function

$$ReLU(x) = \begin{cases} 0, if\ x < 0 \\ x, if\ x \geq 0 \end{cases}$$

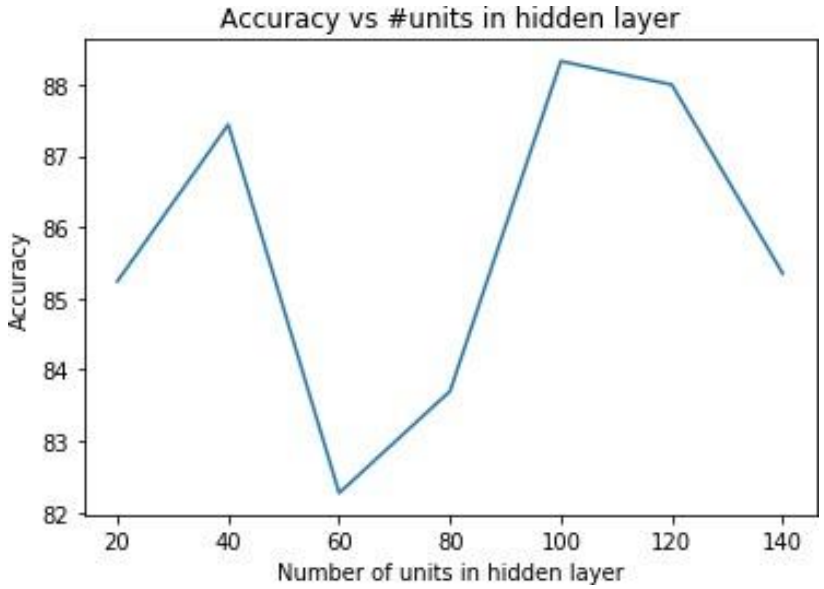
- A Dense hidden layer of 75 neurons and rectifier linear unit activation function
- A Dense output layer of 2 output classes and softmax activation function



Neural Network

To create our fully-connected neural network, we begin with creating a Sequential model, a linear stack of 3 layers (input, hidden and output as mentioned above). Before training the model, we need to configure the learning process, which is done via the compile method. And then we train our model based on our training data, so as to learn our weights.

After trying various numbers of hidden units in the hidden layer, we conclude that the best accuracy is achieved with 100 neurons. However, we choose a hidden layer of 75 neurons since in that situation we have a good percentage and lower execution time.



Accuracy vs Hidden Units in Hidden layer

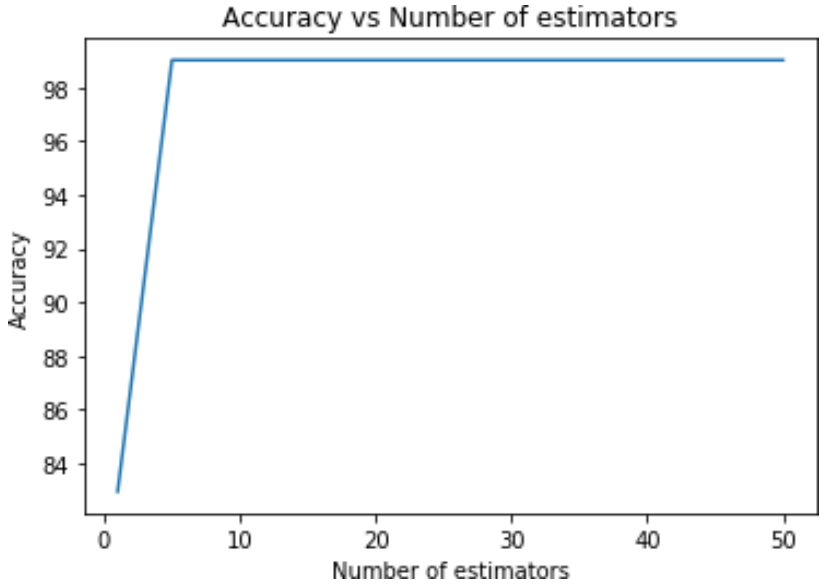
SVM

In our problem, a radial basis function (RBF) kernel is used to train the SVM because such a kernel would be required to address this nonlinear problem.

Large value of parameter C should cause a small margin, and the opposite. Small value of parameter C should cause a large margin. There is no rule to choose a C value, it totally depends on our testing data. The only way is to try different values of the parameters and go with those that gives the highest classification accuracy on the testing phase. As for the gamma parameter, if its value is low then even the far away points can be taken into consideration when drawing the decision boundary and the opposite. With a high gamma parameter, the hyperplane is dependent on the very close points and ignores the ones that are far away from it. So, the best parameters we used in the SVM technique, after testing several variants with trial-and-error approach, were 10 for the C value, and 1 for the gamma value.

RF

The parameters we used after trying several variants (check the figure below), were 5 estimators - the number of trees in the forest – and 5 as the depth of the tree. The algorithm provided us with a high accuracy result in a short time, specifically 99.01% in 0.01 seconds. Generally, our opinion was that the algorithm provided us with the best results in regards to how simple it was to develop and fast to execute.



Accuracy vs Number of estimators (RF)

We can use the RF algorithm for feature importance implemented in scikit-learn. Feature importance refers to a class of techniques for assigning scores to input features to a predictive

model that indicates the relative importance of each feature when making a prediction. After being fit, the model provides a feature importance property that can be accessed to retrieve the relative importance scores for each input feature.

In our model, the first 6 features are the most important and we will achieve relatively great accuracy results even if we do not take into account the rest of the features' set. We can see this in the figure below, where the bar chart has bigger values for these features.

- Number of ports: the number of ports in a circuit. In electrical circuit theory, a port is a pair of terminals connecting an electrical network or circuit to an external circuit, a point of entry or exit for electrical energy.

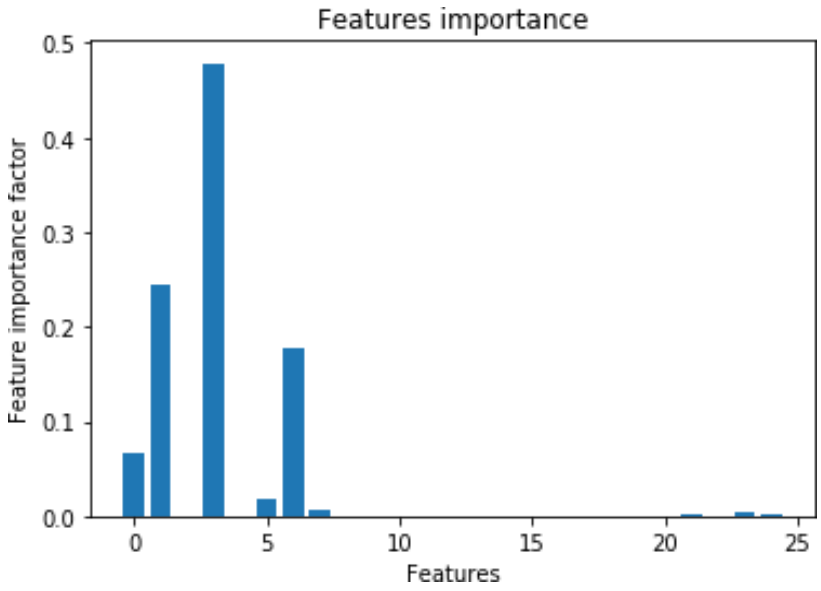
- Number of combinational cells: combinational cells are made up from basic logic NAND, NOR or NOT gates that are "combined" or connected together to produce more complicated switching circuits.

- Number of buf/inv: number of inverter and buffer cells instantiated in the design

- Buf/inv area: area occupied by the above

- Total Cell Area

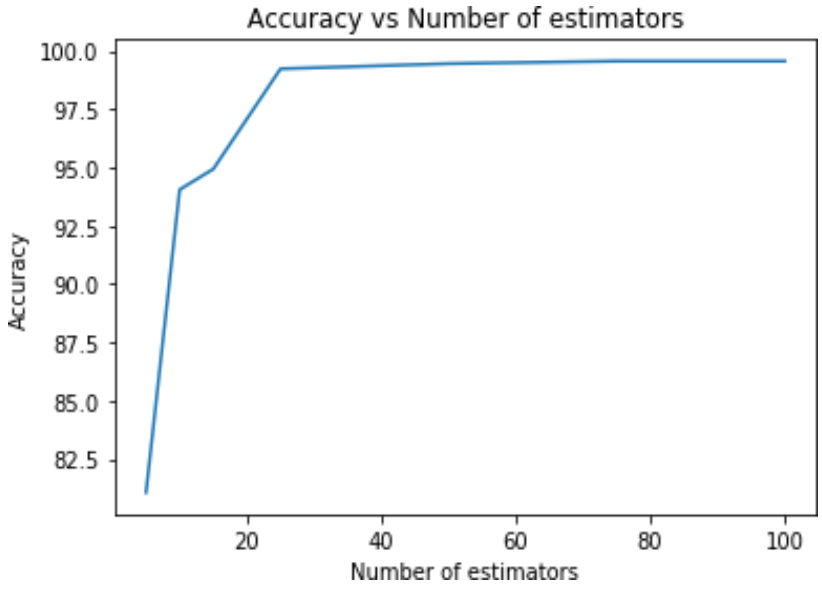
- Total Dynamic Power: Total Power consumed by the design, given switching activity produced by vectors with coverage > 95%



Features importance graph

GB

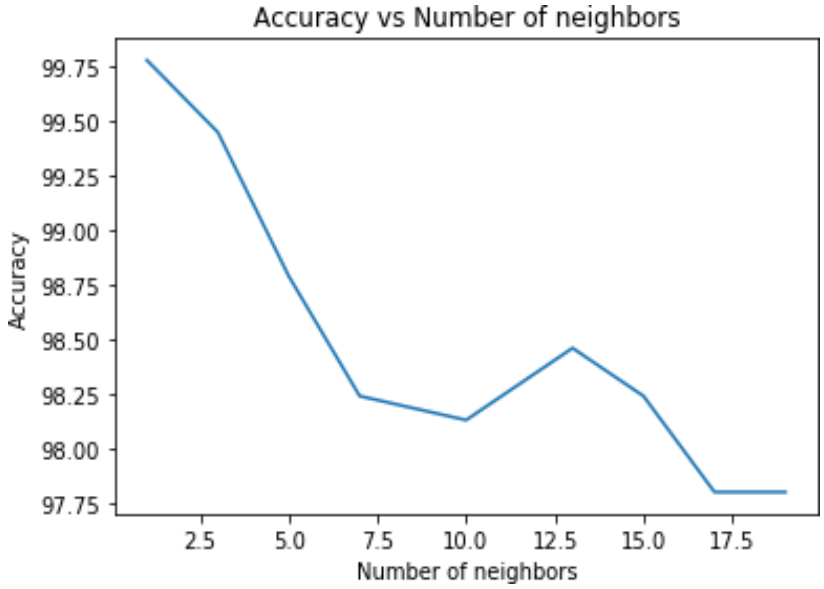
After trying several variants, we set the values 0.1 and 75 (with the smaller execution time) to the learning rate and the estimators - the number of boosting stages to perform – respectively and train our model with 99.56% accuracy.



Accuracy vs Number of estimators (GB)

KNN

In our problem, we choose 3 as the number of neighbors, despite of the fact that we achieve the highest accuracy with 1, since we have better execution time.



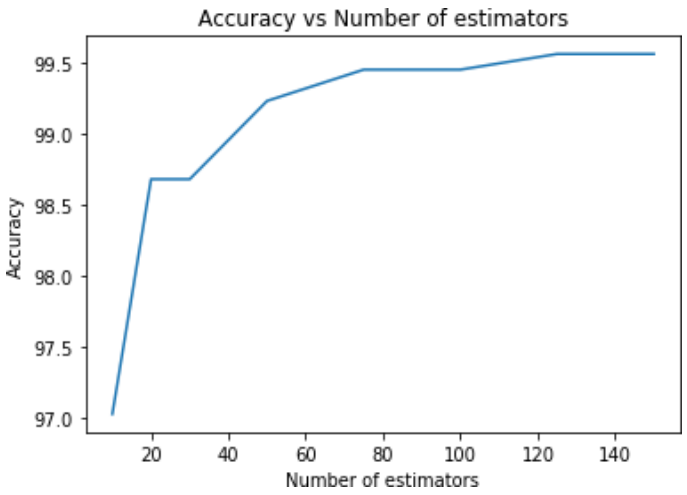
Accuracy vs Number of neighbors

LR

In our dataset, we use binary LR since we have 2 discrete classes for our predicted labels. We use Sklearn's 'LogisticRegression' classifier class, 'liblinear' solver since we have a small dataset and 'or' since the data is binary. In order for the method to converge we choose 300 as max iterations number.

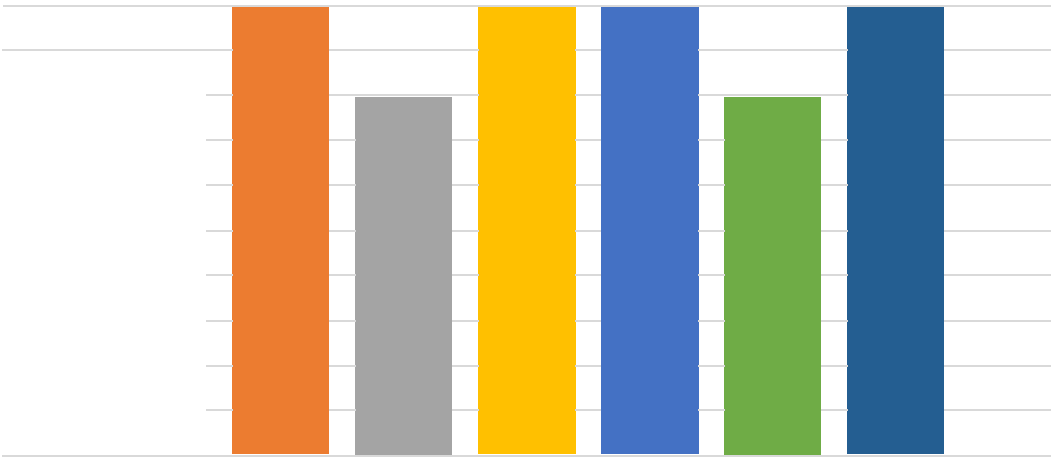
XGB

In our problem, we choose the number of 20 estimators as it provides the highest accuracy in the shortest execution time.



Accuracy vs Number of estimators (XGB)

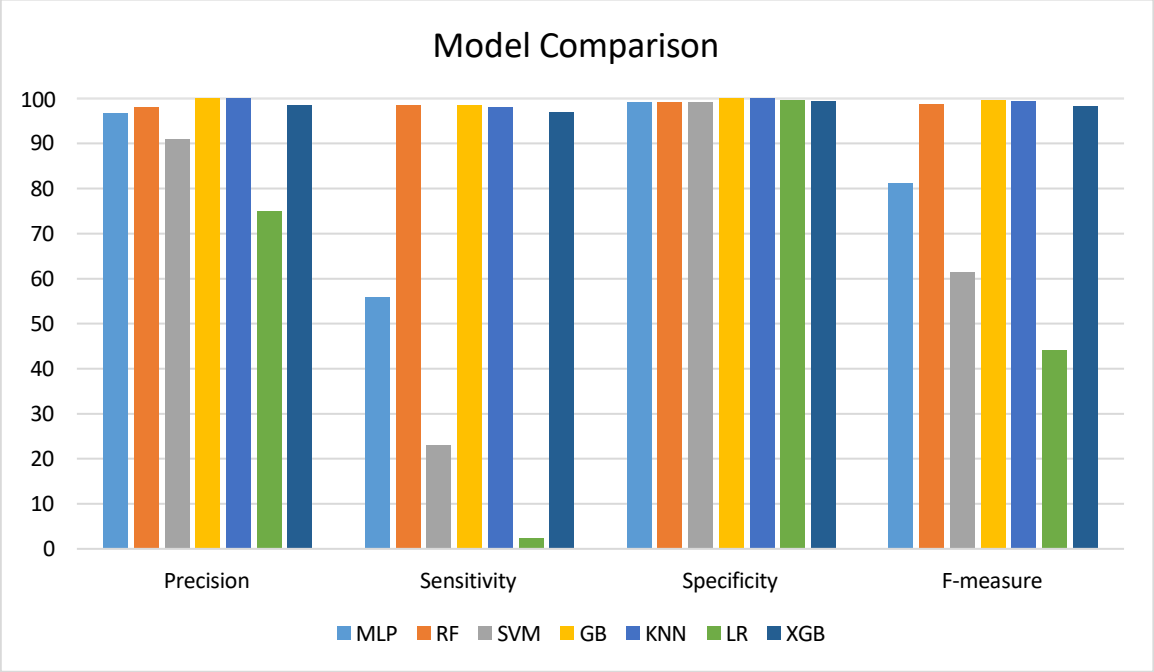
Comparison of the Results



Accuracy comparison of the algorithms (testing-set)

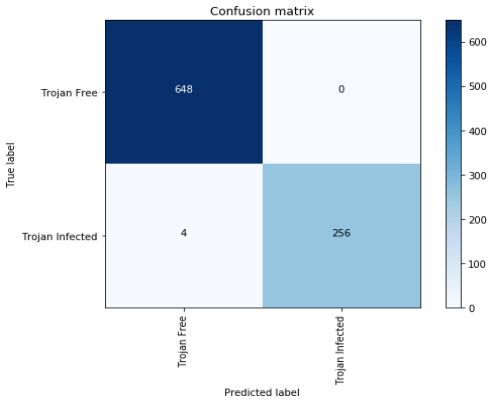
ALGORITHM	ACCURACY	EXECUTION TIME
MLP	84.25 %	4.458 sec
RF	99 %	0.013 sec
SVM	77.3 %	0.217 sec
GB	99.56 %	0.142 sec
KNN	99.45 %	0.084 sec
LR	71.81 %	0.004 sec
XGB	99.45 %	2.706 sec

Model comparison for testing on the testing-set



Metrics comparison of the algorithms (testing-set)

Here, as seen in the figure below GB algorithm did well with classifying the most samples into the correct class. A small percentage was predicted as TI but it was TF in reality.



Confusion Matrix of our Testset

From the above-mentioned metrics and visualizations, it is obvious that GB produces the better results when performed on our training-set.

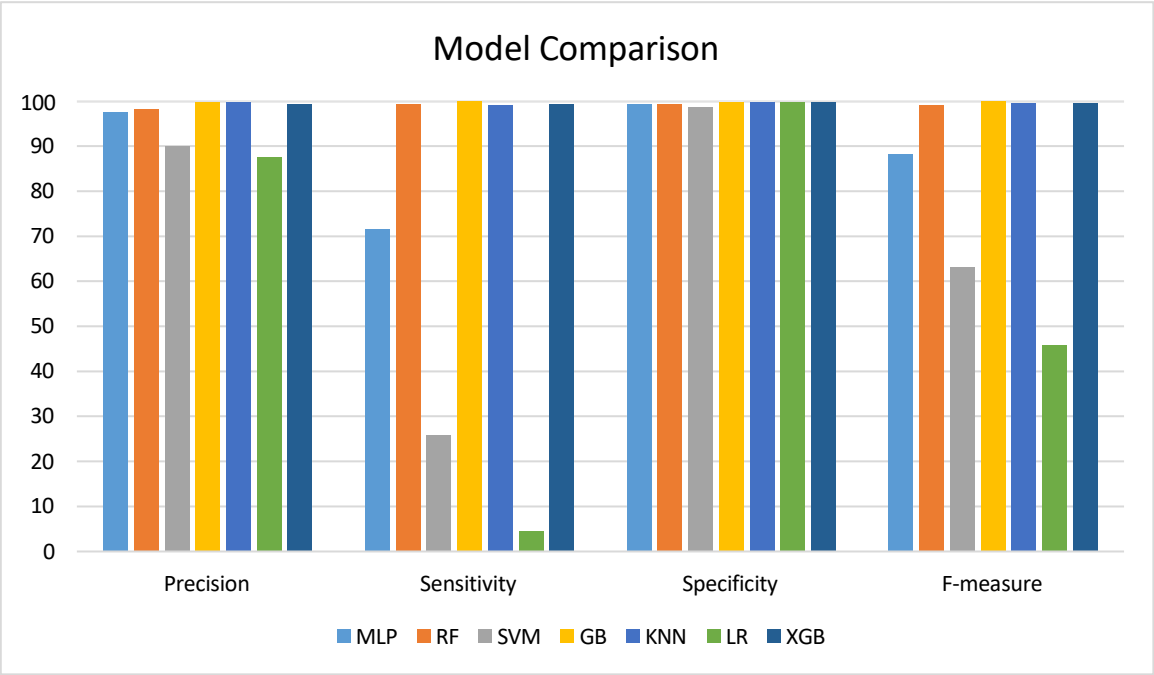
Model comparison on our training-set

Below, we provide a performance comparison between the used algorithms in order to give a better look into which algorithm is considered to be more suitable when we test our models on the training set.

Accuracy comparison of the algorithms (training-set)

ALGORITHM	ACCURACY	EXECUTION TIME	F-measure
MLP	86.11 %	2.209 sec	88.28%
RF	99.29 %	0.009 sec	99.15%
SVM	77.29 %	0.199 sec	63.11%
GB	**99.95 %**	**0.142 sec**	**99.94%**
KNN	99.67 %	0.045 sec	99.61%
LR	71.62 %	0.004 sec	45.86%
XGB	99.62 %	0.304 sec	99.54%

Model comparison for testing on the training-set



Metrics comparison of the algorithms (training-set)

It is obvious that GB produces the better results - on the training set - combining the above-mentioned metrics and visualizations.

Conclusion

Every year, the development rate as well as the sophistication level of HTs are increasing at an alarming rate. Highly elaborate strategies are constantly being introduced for designing HTs that can affect a wide range of ICs, spanning from simple configurations used in most of our everyday life habits to state-of-the-art circuits fabricated for military, industrial and financial purposes or even for state-sponsored sensitive research areas. HTs are stealthy by nature, they come in different form, size and type flavors, while each year HT attacks cost the global economy and mainly the technology companies millions of dollars. If left unchecked, these intrusion attempts will become the most significant obstacle of technological progress in the future, with a severe impact on all aspects of our everyday lives. In this landscape of increased uncertainty related to secure IC design and manufacturing, the need for highly effective, versatile and scalable HT countermeasure methods naturally emerges.

In this thesis in order to identify the HTs, we proposed an HT detection and classification technique via area and power features for GLN phase on ASIC circuits. At first, we design with the Design Compiler NXT software, all the circuits from Trust-HUB library. After that, we developed and compared seven ML models, MLP, RF, SVM, GB, KNN, LR and XGB. We choose the GB model which returned the highest accuracy and F-measure, 100%. And finally, we compared our model with existed state of the art models from journal and conference studies. Our model returned the highest results with average 99.7% accuracy and 99.5 F-measure.

Future Improvements

Future work will include the following improvements. The first one is the ability of our model implementation to classify each sample into a circuit category. In that way, we could find out on which circuit category our methods failed to analyze its features or/and leak information from their values. This can be done by setting the ‘circuit category’ feature as the target value, taking into consideration any subcategories and grouping them to the main ones (eg. 'S38584-T100', 'S38584-T200' and 'S38584-T300' to 'S38584’ team).

The second would make our problem able to deal with real – time data, so we could create our dataset’s samples by inserting HT attacks into an IC. This could be done by having access to this circuit and analyzing the stream of data we passed to it and taking advantage of the already known output and comparing to the real output that the circuit produces. An instant to way to see how a HT can affect the structure of an IC.

Last but not least, a simple improvement could be to expand the number of samples that our datasets contain, while keeping the balance between the number of Hardware Free and Hardware Infected samples. We would have then a better and more representative prediction results.