Machine Learning for a better biological circuit design automation

EC500 D1 - Project Report

Marcia Sahaya Louis
Manuel Giménez
Radhakrishna Sanka

April 25, 2017

1. Motivation

In-silico bio circuits design is an attractive interdisciplinary domain in which engineers designs build tools to generate plasmid as circuits. This approach provides an abstraction and accelerates the design process. But these circuits are sometimes do not behave as expected. This maybe due to inherent high complexity of living organisms and limited by scope of our understanding it. We strongly believe that full understanding of the high-complexity of life is not needed in order to improve the reliability of our designed biocircuits. This deep belief comes from inspecting the history of technology development: for instance humankind was able to build the first arches - key architectural technology - long before understanding much of the phenomena behind them. We still have in our planet, some arches dating from thousands of years, when material science, civil engineering, or even classical mechanics did not exist.

The main question then is: how can we improve our design reliability when it comes to biocircuits? How can we achieve higher rates of correctly working biocircuits in-silico? Our bet is through trial, error and learning. We believe that an explicit automatic learning step has to be included in the traditional Design-Build-Test cycle.

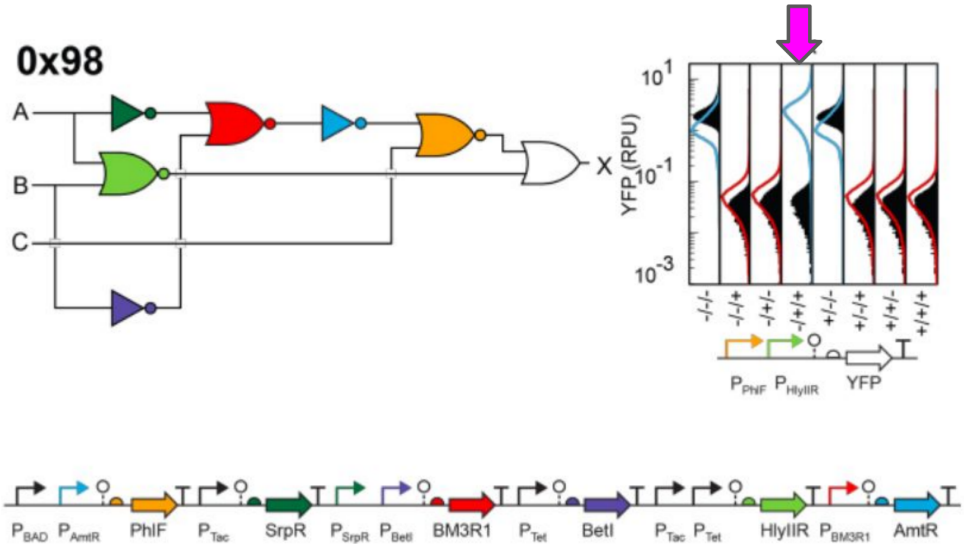

Figure 1

2. Design Tools

2.1 Cello

For this project we focused our attention on Cello, a software tool for for genetic circuit design automation. It generates blueprints of biocircuits that implement the truth table written in Verilog. Biocircuit design generated by Cello are composed of modular pieces taken from highly curated repository of biological not and nor "gates". Cello was used to produce blueprints for 60 logical circuits in [2]. To verify the designs the authors built and tested them in the wetlab. Although obtained results broke success rates previously reported in the literature, around 25% of the designs were not functionally equivalent when implemented in cells as in Figure 1. Our hypothesis is that the underlying model currently used by Cello for circuit assignment is not constrained enough. That is why 25% of the designs proposed by the tool end up computing different boolean functions when implemented in the living organisms. One option would be changing the model and the other would be add a new step feedback loop to the model as in Figure 2 to filter out some combination of gate assignments.

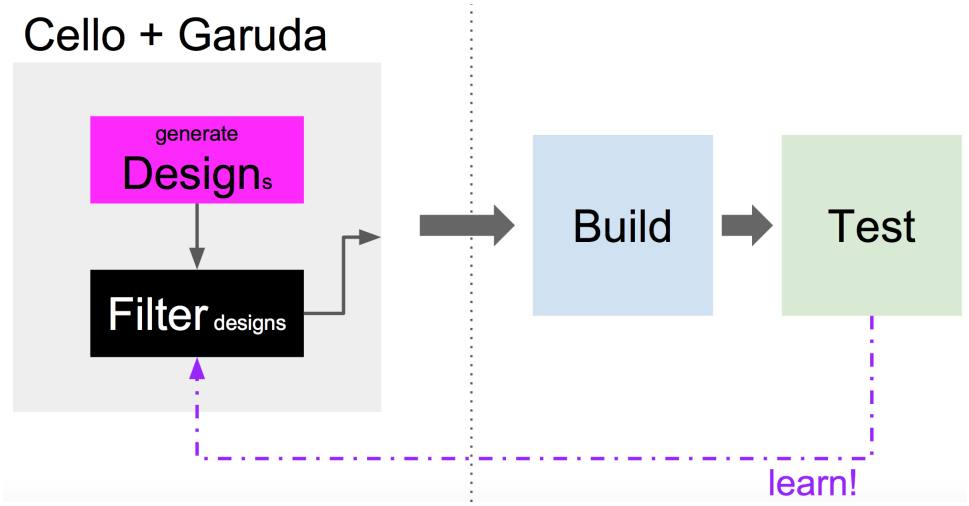

Figure 2

Our objective in this work is to generate models (using machine learning techniques on biocircuit testing data coming from the wetlab) and use them to implement such pre-building filtering step, Figure 2. These will be blackbox "filtering" models that will classify in-silico designs as "defective" or "functional". In machine learning terms, we are framing this new module as a data classifier. These models might not be useful to completely understand the phenome behind a failure or success, but could allow Cello increase its accuracy when generating biocircuits blueprints.

The idea of using machine learning to overcome for enhancing circuit design is an approach also being used in the electronic design automation field. For instance the Center For Advanced Electronics Through Machine Learning (CAEML); they state that [1] "many of the observed failures of integrated circuits during quality testing are the direct result of an insufficient modeling capability". And they propose machine learning as a potential solution to this problem.

3. Machine Learning

We used Support Vector Machines (SVM) for our learning task. SVM is a supervised machine learning algorithm used for classification and regression problems. The model is generated by finding the hyper-plan in d-dimensional feature space that differentiates the two classes. The decision function for the linear model is given by Equation 3.1 and the objective function is given by Equation 3.2. The model is trained such that the objective function is minimized. In our learning task we generated two SVM models, all with C =1.0, Linear kernel, to classify the genetic circuit design as a 'pass' or a 'fail' class. The labels are determined from the paper [2].

$$f(x) = \sum_{i}^{n} \alpha_i y_i (x_i^T x) + b \tag{3.1}$$

$$min_\alpha \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_i y_k (x_j^T x_k) - \sum_i \alpha_i \tag{3.2}$$

subject to

$$0 \leq \alpha_i \leq C \text{ for all } i, \text{ and } \sum_i \alpha_i y_i = 0$$

where,
n number of sample and d is dimension of feature vectors,
α is a n dimensional learning parameters ,

y is the n dimensional true label,x is the n by d training data,
b is the intercept, and C is the penalty for error.

4. Results & Discussion

4.1 Prediction Model

The dataset has 55 circuits used in [2]. Each sample in the dataset has a binary value as the feature vector. A value of '1' if the gate (feature) was used in circuit assignment otherwise '0'. Figure 3 gives an example of feature mapping. To understand the distribution of gate assignment, we plotted the frequency of each gate in the dataset Figure 4. Note that each gate is assigned only one in circuit. It can be seen that some gates are assigned more frequently than others.

We trained three SVM models, the first model has equal weights for all samples data points, the second model has higher weights for class 'fail' and third model with higher weights for samples with class 'fail' labels. Sample/class weighting rescales the C parameter, therefore the classifier puts more emphasis on correctly classifying those data points. When the dataset is unbalanced, it is recommended to use sample/class weighting. Receiver Operating Characteristic

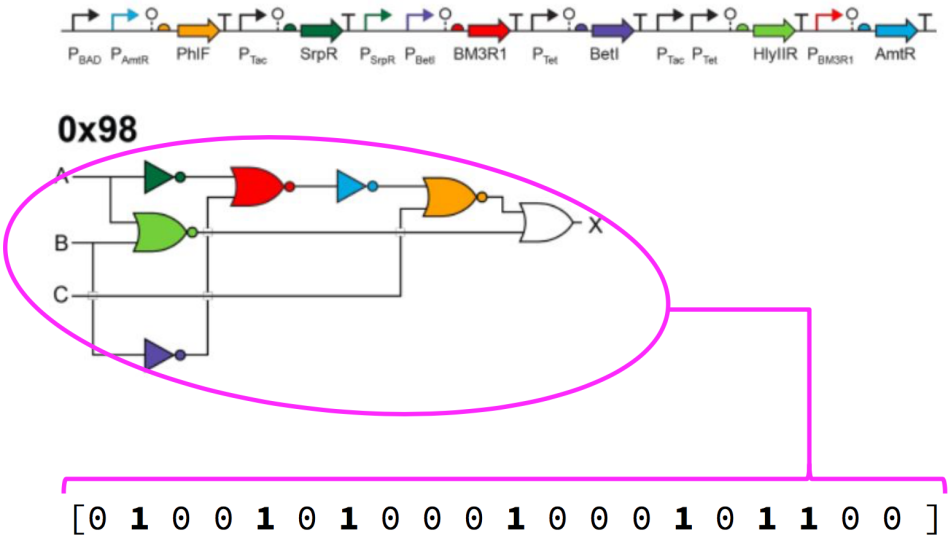

Figure 3

(ROC) is a metric to visualize the classifiers output quality. The curves has true positive rate on the Y axis, and false positive rate on the X axis, as shown in Figure 5. A true positive rate of one and a false positive rate of zero is the ideal point (top left corner of plot) which has maximum Area Under the Curve (AUC). The AUC metric gives a quantitative estimate of classifier output. The model with no class weighting achieves 75% accuracy, model with class/sample weighting achieves 82% for a random split of 60% training and 40% testing data.

To evaluate the accuracy of a classification, we computed the Confusion Matrix. Figure 6 and Figure 7 show the confusion matrix for model with no class weighting and model with class/sample weighting respectively. The model with weighting classifies 'fail' class more reliably. To improve the accuracy of model further, a known technique in the field of machine learning it is to get more data.

In the case of biocircuits, to get more data you need to perform wet lab experiments, something that usually takes a lot of time and effort. We believe that lab automation advancements are key in order to generate more data without such high cost of production. In the long term availability of more data is a condition for our envision to be successful. Another way of improving models in the absence of wetlab data, is using synthetic generated data for training. It is worth noting though

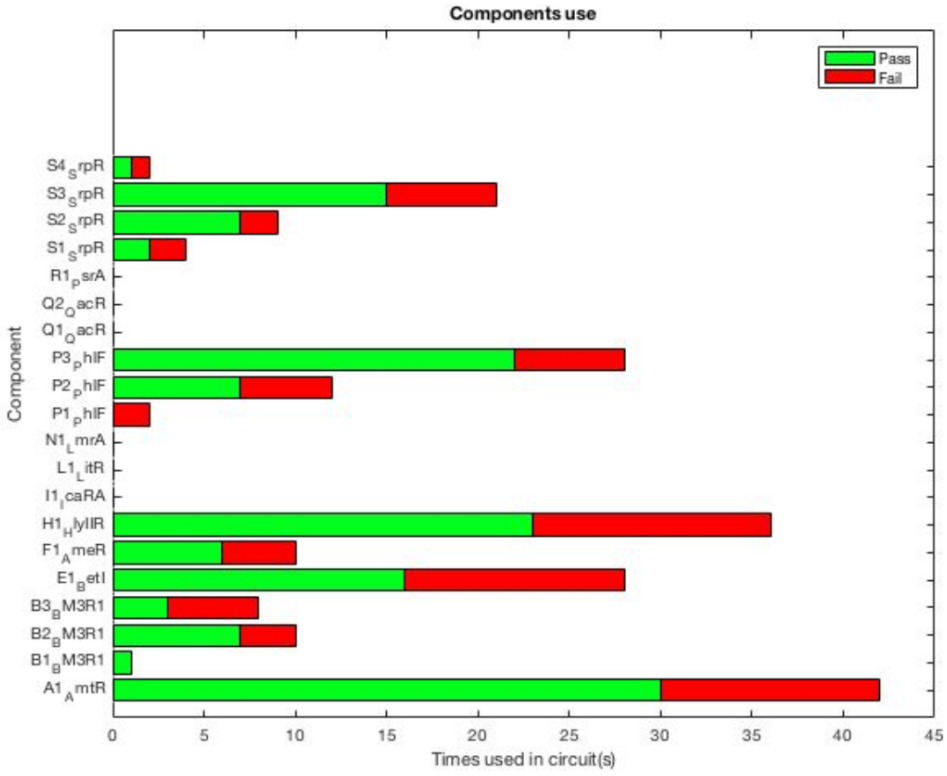

Figure 4

that to do such thing a model that generates "good" synthetic data has to be developed - something that by itself requires either having some insight in the mechanics behind the phenomena taking place, or more raw wetlab data.

4.2 Correctness metric

The correctness metric defined by cosine similarity, is a measure of the direction-length resemblance between vectors. An angle of 0 degrees means that cos = 1 and that the vectors have identical directions; i.e., that the corresponding data sets are completely similar to one another. An angle of 90 degrees means that cos = 0 and that the corresponding variables are perpendicular, but not necessarily that are uncorrelated. The correctness metric can be used to access the degree of variation between boolean function and experimental wetlab results. Currently this result is not integrated with prediction model.

4.3 Software Flow

This research project has 2 software flows that are important to distinguish:

- Model Engineering Flow (Flow A) - This is the flow/tools we develop to generate and develop machine learning models. The software that is built for this flow allows to gather datasets that will allow us to experiment and engineer machine learning models.
- Design Engineering Flow (Flow B)- This is the flow/tools we develop that will utilize the models developed in Flow A. This will be used by engineers who will actually design the biocircuits.

The tool that was developed during the EC500 project is for Flow A. One could say that the Flow A consists of building machine learning infrastructure.

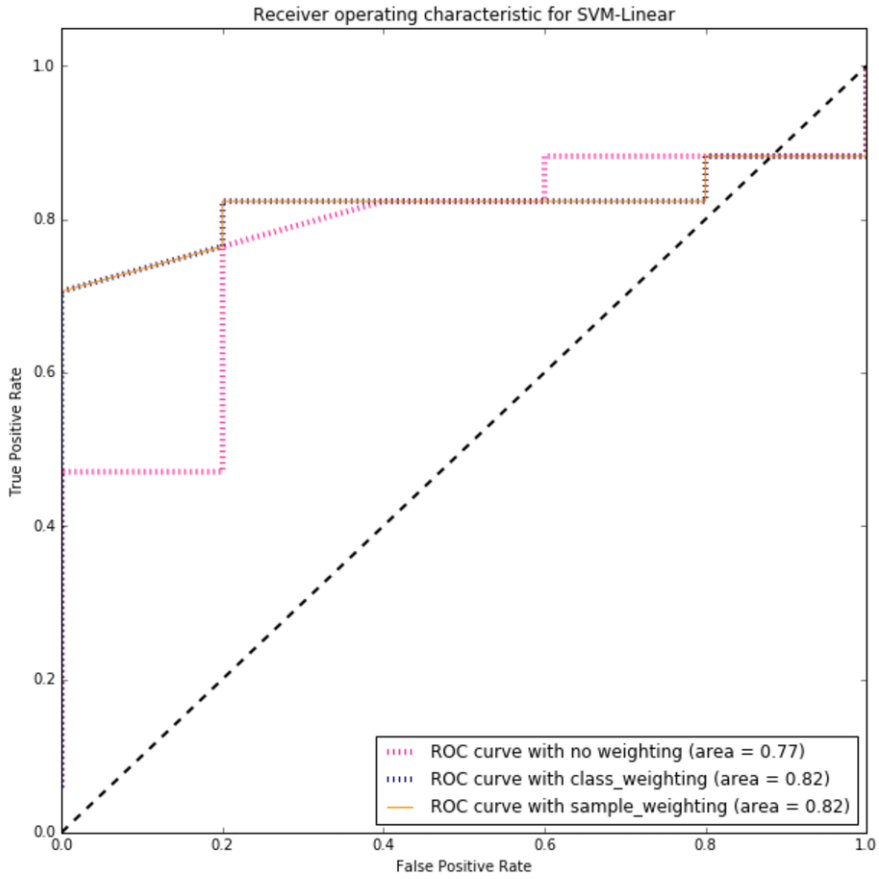

Figure 5

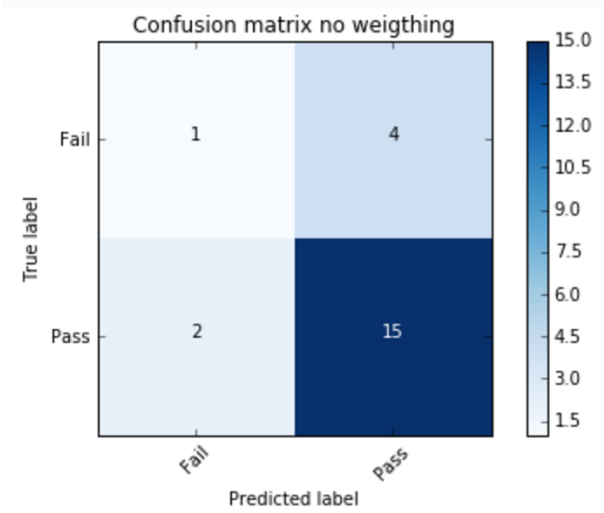

Figure 6

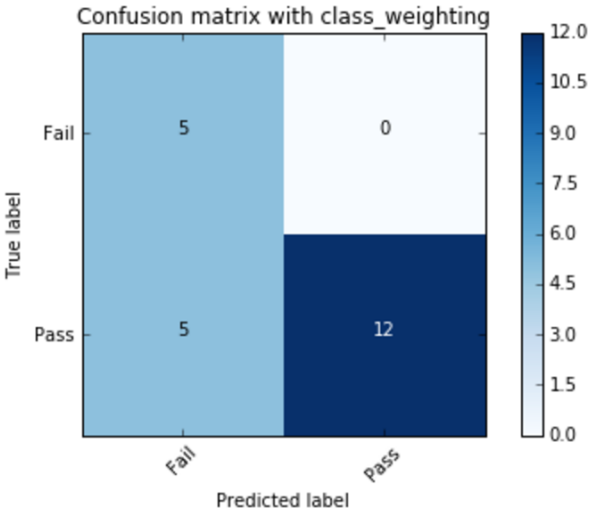

Figure 7

Note: There is a circular dependency between Flow A and Flow B. We are building Flow A because we are still in the experimental stage where we need to figure out what will actually be useful for

4.3.1 Cello Pipeline

The software developed for the purpose of analyzing the Cello circuits is a part of Garuda and is what is termed the "Cello Pipeline". The Cello Pipeline includes both a commandine tool and web-server based system to enable the user to use the GUI. The pipeline consists of the following actions:

1. Predict Pass/Fail for a particular genetic circuit. (Implemented on the backend)
2. Simulate and generate gate assignments for given UCF. (Implemented in the backend)
3. Train the model based on Pass/Fail responses. (Implemented in Python Notebook, need to integrate into the pipeline.)
4.

The following proposed features require an database engine:

1. Upload UCF
2. Store a UCF's Corresponding ML models

The following changes have to be made to Cello to enable the pipeline
Code contributions were made to the following software:

- Cello Adapter - https://github.com/CIDARLAB/CelloAdapter
- Garuda - https://github.com/CIDARLAB/Garuda
- Garuda Python - (Python Notebooks we used for our analysis) https://github.com/CIDARLAB/garuda_python

4.4 Challenges

List of Challenges:

- Trouble interfacing and using cello with cello-engine.
- Road blocks due to lack of reproducibility of few circuit used in Cello paper [2] through cell-engine.
- Limited size of dataset used by machine learning algorithm.

Flagged Issues:

- https://github.com/CIDARLAB/cello/issues/created_by/rkrishnasanka

5. Conclusions

Cello: A Genetic circuit design automation tool has the potential to save wet-lab hours and resources by filtering simulated circuits by using Cello's existing constrains and additional using the prediction model. Further by training with more data and adding more features unique to gate property besides the mere presence or absence mapping, we expect to improve accuracy of the model. Inspired from EDA, we will be generating various test vectors using generative machine learning algorithms to increase robustness of Cello.

References

[1] Center for advanced electronics through machine learning caeml.

[2] Alec AK Nielsen, Bryan S Der, Jonghyeon Shin, Prashant Vaidyanathan, Vanya Paralanov, Elizabeth A Strychalski, David Ross, Douglas Densmore, and Christopher A Voigt. Genetic circuit design automation. *Science*, 352(6281):aac7341, 2016.