[[1]](#footnote-2)

Predicting Performance of Integrated Circuits using Machine Learning

Joanna Duran ([joannad@mail.smu.edu](mailto:joannad@mail.smu.edu)), Nikhil Gupta ([guptan@smu.edu](mailto:guptan@smu.edu)), Max Moro ([mmoro@smu.edu](mailto:mmoro@smu.edu))

**Abstract — Semiconductor manufacturing is a variable process and outcomes depend on several factors. To meet target specifications, some parameters are controlled by design engineers. However, many parameters are beyond human control (e.g. process variation). The output variables that are measured after the manufacturing process is complete must fall within a specified range of values (target specification). Variation in the manufacturing process may lead to issues if the outputs are outside the minimum or maximum value of these specifications. Through this work, we aim to build a model that can be used to predict the performance of an integrated circuit. This model could be used to preemptively take actions to prevent specification violation after manufacturing.**

*Index Terms – Semiconductors, Integrated Circuits, Predictive Modeling, Machine Learning, Feature Engineering, Linear Regression, Principal Component Analysis (PCA), LASSO, LARS, Quadratic Discriminate Analysis (QDA), Logistic Regression.*

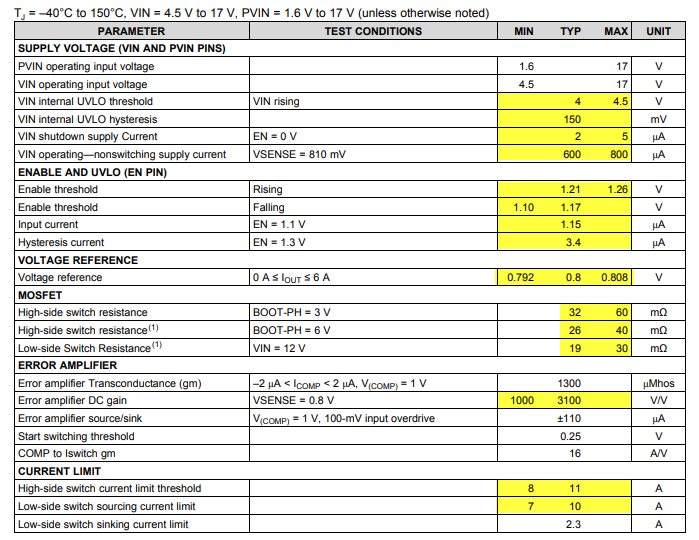
# INTRODUCTION

L

ike in most manufacturing environments, the question that is often asked is “Can we predict the performance before the device is manufactured and preemptively make changes when the output is expected to be outside the desired range?” The semiconductor manufacturing, the answer is yes. Current practice is to use electrical simulation (plus running Monte Carlo simulations) to identify specification limits. However, this is very resource and time intensive as each electrical simulation can take several hours to run.

Figure 1 shows an example specification sheet for an integrated circuit. Each row is a single output with its respective minimum, typical and maximum measured values. Note however that not all values are populated. This may be due to several reasons including but not limited to time and cost constraints to measure this in hardware.

The objective for this project was to build an accurate model that can be used to predict the performance (min, typical, max values) of an integrated circuit. This model could be useful to preemptively take action to make sure the measured output is within specification limits after manufacturing. In other cases, this model could also be used to predict the limits in cases where it is time and cost prohibitive to measure this on hardware. A target accuracy of ±10% was desired from this model, but ±15% was also acceptable if these occurrences were rare.



*Fig 1: Sample output from an integrated circuit* [*Reference*](http://www.ti.com/lit/ds/symlink/tps54620.pdf)

# Literature Review

This project is a continuation of Project 1 for course DS6372 “Applied Statistics”, Spring 2019 [**XX**]. The linear regression models obtained from project 1 suffered from assumption violations, the most severe being the equal variance assumption. The normality of the residuals was also a concern since they were right skewed, although this violation was not severe due to the large sample size. It was also observed that 2-way interaction of all features was not practical without dimensionality reduction since it would create roughly 28,680 predictors with only 6980 observations. Another interesting observation was that a few observations in the training set were high leverage points and this may have impacted the accuracy of the final model.

From project 1, we identified a few possible improvements that could be made to the model fit. These improvements pointed to the need for either (1) performing intelligent/selective feature engineering and variable selection using domain expertise, or (2) building a model to predict high influence points from the available training data, predicting if a new observation belongs to this group and finally, using this categorical prediction as an additional input to the eventual linear regression model, or (3) using non-parametric models such as tree-based models. In this project, we acted upon the first idea and touched upon the second one to some extent.

# Data Description and Collection

Data for this project was sponsored by Texas Instruments Inc. (TI). Due to proprietary nature of the information, the variables were anonymized. The true identity of the variables was known to only one of the authors of this paper through their association with TI. This information was important as it was used to do the selective feature engineering that will be discussed later in this paper.

The data consisted of 10,000 observations capturing the performance of an integrated circuit under various conditions. There were 240 features consisting of:

1. Engineer-controlled variables (x1 – x23). Values for these variables were spread across a large range; some were in the range of 1 to 100 while others were in the Nano or Micro range.
2. Process variation variables (stat1 – stat217). These parameters are beyond human control. They represent various statistical manufacturing parameters. The variables varied between -3 and 3 representing the ±3 sigma variation around the mean (typical) process.
3. Output Variables (y1 - y19) which represented various output variables.

The engineer-controlled variables have a predefined range of values that the engineer can chose from. Since they can pick any value in this range, the values for these variables were uniformly and randomly sampled from the range of acceptable values while the data was being collected. Statistical features were also uniformly randomly sampled since the goal was to obtain good model accuracy throughout the statistical variation range and not just closer to the population means (which would have been the case if the data was sampled using a gaussian distribution since in that case, the training data would have had more points closer to the mean and very few points at the ± 3 sigma level).

# Research Methodology

## Output Selection

After discussion with subject matter experts from Texas Instruments, it was decided to focus on modeling ‘y3’ as it was a critical output of this integrated circuit.

## Modeling Approaches

For this project, we tried 2 different approaches. The first one involved performing intelligent feature engineering using semiconductor domain expertise. This included looking at theoretical equations from semiconductor theory and creating the necessary variables from the ones that were available. Once these variables were created, we performed 2-way interaction of all the variables. As discussed, earlier, this leads to more predictors than observations. Hence, we followed this with dimensionality reduction using Principal Component Analysis (PCA). The reduced Principal Components (PCs) were then used to build a linear regression model. First a full model was created using the filtered PCs and this was followed by variable selection using LASSO and LARS. Forward, Backward and Stepwise selection were not used in this project because these algorithms were taking a long time to run due to the large number of predictors even after dimensionality reduction.

In the second approach, we build the initial steps of a model pipeline. We used the full model from project 1 to label the training observations as high leverage or low leverage. This data was then used to train various classification models including Linear/Quadratic Discriminate Analysis (LDA/QDA) and Logistic Regression models. The first step of the proposed pipeline involved classifying new observations into 2 levels, high leverage or low leverage, using the classification models built. Once classified, the predicted class level could then be used as a predictor for the second and final stage of the pipeline which would be the linear regression model for predicting y3. The hope was that by introducing this categorical predictor in the linear regression model, we would intrinsically build 2 separate regression lines – one each for the high and low leverage points and that this would lead to better prediction accuracy.

## Good Modeling Practices

When building the predictive models, we wanted to avoid overfitting the training dataset. This was especially critical since we were expanding our predictors using 2-way interactions of all variables. Even though we eventually follow this with PCA which reduces the number of predictors to XX, we are still at risk of overfitting with such huge number of predictors. To avoid overfitting, we used an 80:20 ratio split on our dataset to develop the Train and Test sets. In addition, during the training process, a 10-fold cross validation technique was used for model selection.

# Exploratory Data Analysis

## Data Preparation

Like most real-world datasets, this one also needed some cleaning. Basic descriptive statistics revealed that 3020 NA values were present for y3 (Fig 2). After consulting with the expert from TI, we found that the predictors (features) for these data points were not practical in combination with each other. Hence, these points are not valid and could be removed without impacting the predicting power of the model being developed.

message('Original cases: ',nrow(data.ori))

## Original cases: 10000

message('Non-Complete:', nrow(data.notComplete))

## Non-Complete: 3020

*Fig 2: Original data and observations with missing values*

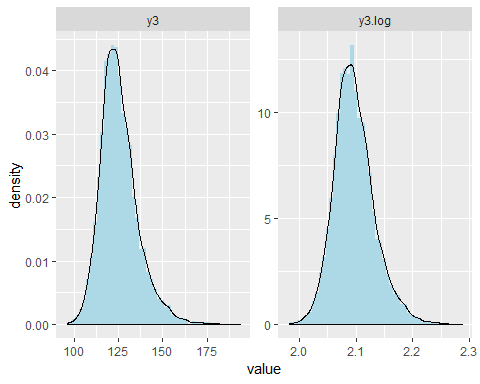
Note that if these were indeed valid data points, we could not have simply removed them from the dataset since it would have violated the random sampling we performed initially, and this would have affected the generalization of the model to the entire design space (population).

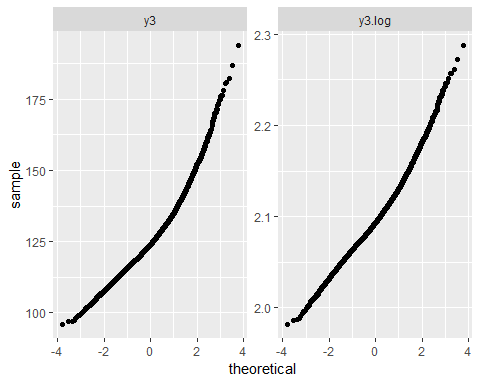
## Output

We begin exploratory data analysis on the target variable y3 and notice right skewness, therefore we perform a log transformation. Log transformation makes the data a little more normal, therefore we proceeded with the log transformed variable “y3.log” (Fig 3). It is important to note that base 10 is more common in this industry, which is why it was used instead of natural log.

## Input Predictors

VIF and Correlations: To determine if there is a correlation within predictors (features), we began by checking for multicollinearity. Since inputs were randomly selected, we did not expect there to be multicollinearity. After running the analysis, the VIF values (Fig 4) confirmed that there was no issue with multicollinearity.



**

*Fig 3: Histogram and QQ plot of y3 and log(y3)*

## Variables VIF

## 1 stat202 1.063592

## 2 stat141 1.062435

## 3 stat52 1.062123

## 4 stat178 1.062030

## 5 stat164 1.059900

## 6 stat184 1.059400

## 7 stat70 1.058888

## 8 stat150 1.058825

## 9 stat14 1.058728

## 10 stat37 1.058385

*Fig 4: Top 10 predictors by VIF*

## Transformations

Talk about the intelligent feature engineering done here. Was transformation done on x18 (sqrt)?

# Model Results

We used an XX% cutoff threshold for cumulative explained variance in the predictors. This cut down the number of principal components to XX.

## Hardware & Library Installation

We need to test the two languages over a wide variety of test

### Virtual Machine (VM) running on a local machine.

The VM is managed by Oracle VM Virtual Box software, version 5.2.26 r128414 (Qt5.6.2). The VM has been.

### Single Node AWS Machine (using Databricks)

Databricks is a cloud platform that provides a unified framework for data processing. The main advantage of using Databricks is the immediate availability of Scala, Py

# Discussion

Table 1 shows the results of the queries run in PySpark and Scala using the Single Node AWS Machine configuration for a 10MB dataset.

The preliminary findings show that the time taken to perform the various queries is roughly the same for both PySpark and Scala (with Scala even being slower than PySpark in many cases).

TABLE I

Preliminary Results – 10MB Data Set

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Query | PySpark Time (sec) | | | Scala Time (sec) | | | Difference PySpark - Scala (sec) | | |
| *Rows Operations* | | |  | | |  | | |  | | |
|  | | Filter Based On Value | 0.69 | | | 0.94 | | | -0.26 | | |
|  | | Filter Regular Expression 1 | 0.54 | | | 0.66 | | | -0.12 | | |
|  | | Filter Regular Expression 2 | 0.47 | | | 0.94 | | | -0.47 | | |
|  | | Shift (Lag) | 1.31 | | | 0.83 | | | 0.48 | | |
|  | | Running Sum | 1.34 | | | 0.65 | | | 0.68 | | |
|  | | Writing 100 New Rows | 0.86 | | | 0.98 | | | -0.12 | | |
|  | | Writing 1,000 New Rows | 0.84 | | | 0.84 | | | 0.00 | | |
|  | | Writing 10,000 New Rows | 1.44 | | | 1.19 | | | 0.24 | | |
| *Column Operations* | | |  | | |  | | |  | | |
|  | | Full Outer Join 3 Columns | 14.93 | | | 0.68 | | | 8.26 | | |
|  | | Full Outer Join 5 Columns | 6.26 | | | 13.67 | | | -7.41 | | |
|  | | Full Outer Join 10 Columns | 5.78 | | | 5.89 | | | -0.11 | | |
|  | | Left Outer Join 3 Columns | 5.00 | | | 3.77 | | | 1.23 | | |
|  | | Left Outer Join 5 Columns | 2.41 | | | 2.28 | | | 0.12 | | |
|  | | Left Outer Join 10 Columns | 2.19 | | | 2.33 | | | -0.14 | | |
|  | | Inner Join 3 Columns | 8.89 | | | 3.47 | | | 5.42 | | |
|  | | Inner Join 5 Columns | 2.87 | | | 2.36 | | | 0.52 | | |
|  | | Inner Join 10 Columns | 2.11 | | | 2.44 | | | -0.33 | | |
|  | | Sorting Ascending 1 Column | 1.86 | | | 2.41 | | | -0.55 | | |
|  | | Sorting Ascending 5 Columns | 1.92 | | | 2.48 | | | -0.55 | | |
|  | | Sorting Ascending 10 Columns | 1.98 | | | 2.40 | | | -0.42 | | |
|  | | Sorting Descending 1 Column | 1.74 | | | 1.98 | | | -0.24 | | |
|  | | Sorting Descending 5 Columns | 2.90 | | | 2.14 | | | 0.76 | | |
|  | | Sorting Descending 10 Columns | 4.92 | | | 2.22 | | | 2.71 | | |
|  | | Merge 2 Columns in 1 | 1.23 | | | 0.83 | | | 0.40 | | |
|  | | Merge 5 Columns in 1 | 1.11 | | | 0.91 | | | 0.19 | | |
|  | | Merge 10 Columns in 1 | 1.16 | | | 1.53 | | | -0.37 | | |
|  | | Split 1 Column in 5 | 1.29 | | | 1.15 | | | 0.14 | | |
|  | | Split 1 Column in 10 | 2.19 | | | 1.18 | | | 1.01 | | |
|  | | Mathematical Operations | 0.47 | | | 0.61 | | | -0.14 | | |
| *Aggregate Operations* | | |  | | |  | | |  | | |
|  | | GroupBy 1 Column | 9.90 | | | 12.62 | | | -2.72 | | |
|  | | GroupBy 5 Column | 13.73 | | | 12.61 | | | +1.12 | | |
|  | | GroupBy 10 Columns | 13.25 | | | 13.63 | | | -0.38 | | |
|  | | Ranking by Group | 2.53 | | | 2.60 | | | -0.08 | | |
| *Mixed Operations* | | |  | | |  | | |  | | |
|  | | Pivot 1 Row and 1 Column | 3.51 | | | 3.60 | | | -0.09 | | |
|  | | Pivot 5 Rows and 1 Column | 4.81 | | | 10.52 | | | -5.71 | | |
|  | | Pivot 10 Rows and 1 Column | 5.92 | | | 8.61 | | | -2.69 | | |
| *Total* | | | |  | | |  | | |  | | |
|  | | *Average Time* | *3.16* | | | *3.72* | | | *-0.56* | | |

# Conclusion and Next Steps

Steps for the remaining part of the Project. Blah bsaaha ajbkacka ahka aclahl

# References

|  |  |
| --- | --- |
| [1] | "Hadoop MapReduce Tutorial," [Online]. Available: https://hadoop.apache.org/docs/r1.2.1/mapred\_tutorial.html. |
| [2] | Zaharia, Matei, et al., "Fast and interactive analytics over Hadoop data with Spark," Usenix Login, 2012. |
| [3] | Spark, "Spark API Documentation," [Online]. Available: https://spark.apache.org/docs/2.3.1/api.html. |
| [4] | P. Gandhi, "Apache Spark : Python vs. Scala," KDnuggets, May 2018. [Online]. Available: https://www.kdnuggets.com/2018/05/apache-spark-python-scala.html. |
| [5] | Nti-Addae, Yaw; et al., "Benchmarking Database Systems for Genomic Selection Implementation," bioRxiv, 2019. |
| [6] | N. Gureev, "Hive, Spark, Presto for Interactive Queries on Big Data," KTH Royal Institute of Technology, Stockholm, 2018. |
| [7] | S. Akhadov, "PySpark at Bare-Metal Speed," ETH, Zurich, 2017. |
| [8] | Databricks, "Databricks FAQ," Databricks, 2018. [Online]. Available: https://databricks.com/product/faq. [Accessed 15 03 2019]. |
| [9] | DWYL, [Online]. Available: https://github.com/dwyl/english-words. [Accessed 10 03 2019]. |

1. [↑](#footnote-ref-2)