# Machine Learning Engineer Nanodegree

## Capstone Report

Manivassakam Mouttayen
December 28th, 2017

# I. Definition

## Project Overview

The computerization of industrial machinery is increasing – it's becoming equipped with numerous sensors and connected to the rest of the factory via Internet of Things (IoT) platforms. Semiconductor manufacturing is one of the most technologically complex and expensive process with several complex electronic equipment in the line being monitored by sensors.

A major problem faced by businesses in asset-heavy industries such as manufacturing is the significant costs that are associated with delays in the production process due to yield problems. Most of these businesses are interested in predicting these problems in advance so that they can proactively prevent the problems before they occur which will reduce the costly impact caused by downtime

Modern manufacturing technology is starting to incorporate machine learning throughout the production process. Predictive algorithms are being used to plan machine maintenance adaptively rather than on a fixed schedule. Meanwhile, quality control is becoming more and more automated, with adaptive algorithms that learn to recognize correctly manufactured products and reject defects [1]. In this capstone I have applied Machine Learning Techniques to predict failure in Semiconductor Manufacturing Process and thereby help improve the yield.

The project uses the SECOM [4] dataset from the University of California at Irvine's machine learning database. This dataset from the semiconductor manufacturing industry which provides sensor values and also a label which saw if that batch of chip has passed or failed. We will use this data to develop a predictive model that learns and tries to find if a certain set of sensor values are a good indicator of the chip Pass/Fail criteria.

## Problem Statement

In this project we propose machine learning techniques to automatically generate an accurate predictive model to predict faults during the wafer fabrication process of the semiconductor industries. The objective is to construct a decision model to help detect as quickly as possible any equipment faults in order to maintain high process yields in manufacturing. The sensor values are used as an indicator of a potential problem which can then be quickly fixed and the yield improved.

The problem will achieve the following two objectives.
• Build a model that predicts yield failure on their manufacturing process, and
• Through the analysis determine the factors that lead to yield failures in their process.

The approach we will take is to develop a binary classification model for modelling the semiconductor test PASS/Fail condition. The most popular binary classification model is the Logistic Regression. We will use this as our baseline model. Decision tree based models are known to provide a higher accuracy than the baseline model. Hence, we will also consider a decision tree ensemble technique called Random Forest. One of the most important data preprocessing technique in any classification is to ensure that the data is balanced. Often class imbalance leads to poor predictive models. This problem will be addressed by oversampling of the minority class using KNN algorithm.

This is a very important business problem for semiconductor manufacturers since their process can be complex, involving involves several stages from raw sand to the final integrated circuits. Given the complexity, there are several factors that can lead to yield failures downstream in the manufacturing process. Identifying the most important factors helps process engineers improve the yield, and reduce error rates and cost of production, leading to increased productivity.

## Metrics

The main metric used for classification problems is the classification accuracy. However, for this problem which has call imbalance high accuracy will often be misleading. Here Recall will be a more appropriate metric [3].

The SECOM dataset we use has only 104 cases of yield failure out of 1,567 examples. This is a 6.6% failure rate. With such an imbalanced class even, a naïve model can show high accuracy by simply predicting “yield pass” every time. Such a model would have a 93.4% accuracy without correctly identifying a single case of yield failure. With this class imbalance the model learns to predict which cases will pass but fails to learn about yield failures which is the goal of the project.

Recall is the percentage of yield failures that the model identifies which matches with the goals of the project and hence will be used as a metric to evaluate the model.

The metrics can be computed with the help of a confusion Matrix.

## Confusion Matrix

A confusion matrix shows the number of correct and incorrect predictions made by the classification model compared to the actual outcomes (target value) in the data. The matrix is *N*x*N*, where *N* is the number of target values (classes). Performance of such models is commonly evaluated using the data in the matrix. The following table displays a 2x2 confusion matrix for two classes (Positive and Negative).

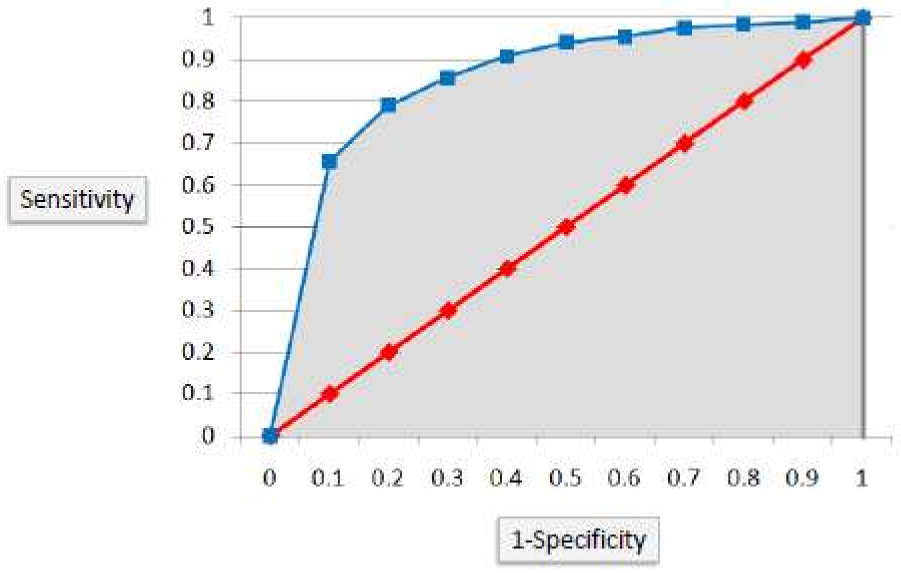
- **Accuracy:** the proportion of the total number of predictions that were correct.
- **Positive Predictive Value** or **Precision:** the proportion of positive cases that were correctly identified.
- **Negative Predictive Value**: the proportion of negative cases that were correctly identified.
- **Sensitivity** or **Recall:** the proportion of actual positive cases which are correctly identified.
- **Specificity:** the proportion of actual negative cases which are correctly identified.

| Confusion Matrix | | **Target** | | | |
|---|---|---|---|---|---|
| | | Positive | Negative | | |
| **Model** | Positive | a | b | *Positive Predictive Value* | a/(a+b) |
| | Negative | c | d | *Negative Predictive Value* | d/(c+d) |
| | | *Sensitivity* | *Specificity* | **Accuracy** = (a+d)/(a+b+c+d) | |
| | | a/(a+c) | d/(b+d) | | |

## Area Under the Curve (AUC)

Area under ROC curve is often used as a measure of quality of the classification models. A random classifier has an area under the curve of 0.5, while AUC for a perfect classifier is equal to 1. In practice, most of the classification models have an AUC between 0.5 and 1

An area under the ROC curve of 0.8, for example, means that a randomly selected case from the group with the target equals 1 has a score larger than that for a randomly chosen case from the group with the target equals 0 in 80% of the time. When a classifier cannot distinguish between the two groups, the area will be equal to 0.5 (the ROC curve will coincide with the diagonal). When there is a perfect separation of the two groups, i.e., no overlapping of the distributions, the area under the ROC curve reaches to 1 (the ROC curve will reach the upper left corner of the plot).

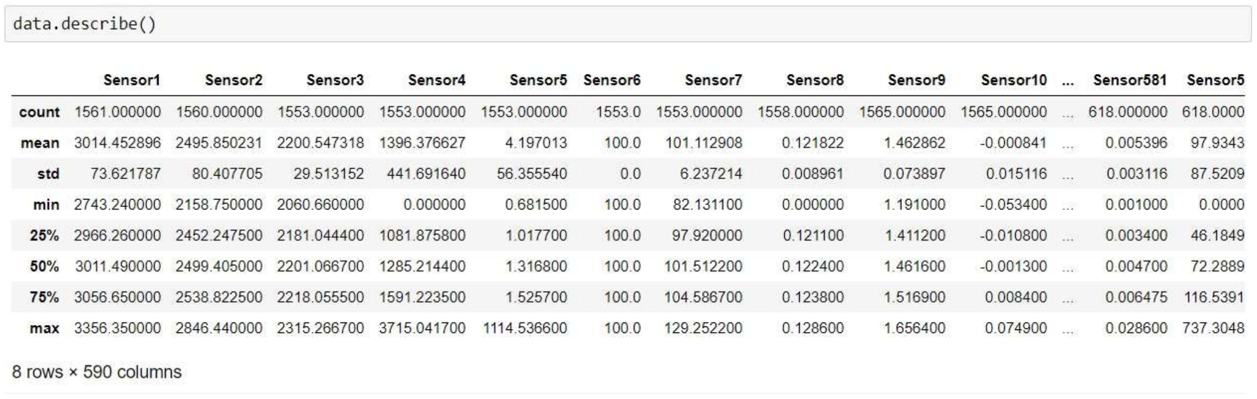


# II. Analysis

## Data Exploration

The input data uses the SECOM [4] dataset from the University of California at Irvine's machine learning database. This dataset from the semiconductor manufacturing industry was provided by Michael McCann and Adrian Johnston. The SECOM data set is available at the University of California at Irvine's Machine Learning Repository.

The dataset contains 1,567 observations, each with 591 sensor readings monitoring a manufacturing line. Of the 1,567 observations, 104 of them represent yield failures.



```
data.describe()
```

| | Sensor1 | Sensor2 | Sensor3 | Sensor4 | Sensor5 | Sensor6 | Sensor7 | Sensor8 | Sensor9 | Sensor10 | ... | Sensor581 | Sensor5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 1561.000000 | 1560.000000 | 1553.000000 | 1553.000000 | 1553.000000 | 1553.0 | 1553.000000 | 1558.000000 | 1565.000000 | 1565.000000 | ... | 618.000000 | 618.0000 |
| **mean** | 3014.452896 | 2495.850231 | 2200.547318 | 1396.376627 | 4.197013 | 100.0 | 101.112908 | 0.121822 | 1.462862 | -0.000841 | ... | 0.005396 | 97.9343 |
| **std** | 73.621787 | 80.407705 | 29.513152 | 441.691640 | 56.355540 | 0.0 | 6.237214 | 0.008961 | 0.073897 | 0.015116 | ... | 0.003116 | 87.5209 |
| **min** | 2743.240000 | 2158.750000 | 2060.660000 | 0.000000 | 0.681500 | 100.0 | 82.131100 | 0.000000 | 1.191000 | -0.053400 | ... | 0.001000 | 0.0000 |
| **25%** | 2966.260000 | 2452.247500 | 2181.044400 | 1081.875800 | 1.017700 | 100.0 | 97.920000 | 0.121100 | 1.411200 | -0.010800 | ... | 0.003400 | 46.1849 |
| **50%** | 3011.490000 | 2499.405000 | 2201.066700 | 1285.214400 | 1.316800 | 100.0 | 101.512200 | 0.122400 | 1.461600 | -0.001300 | ... | 0.004700 | 72.2889 |
| **75%** | 3056.650000 | 2538.822500 | 2218.055500 | 1591.223500 | 1.525700 | 100.0 | 104.586700 | 0.123800 | 1.516900 | 0.008400 | ... | 0.006475 | 116.5391 |
| **max** | 3356.350000 | 2846.440000 | 2315.266700 | 3715.041700 | 1114.536600 | 100.0 | 129.252200 | 0.128600 | 1.656400 | 0.074900 | ... | 0.028600 | 737.3048 |

8 rows × 590 columns

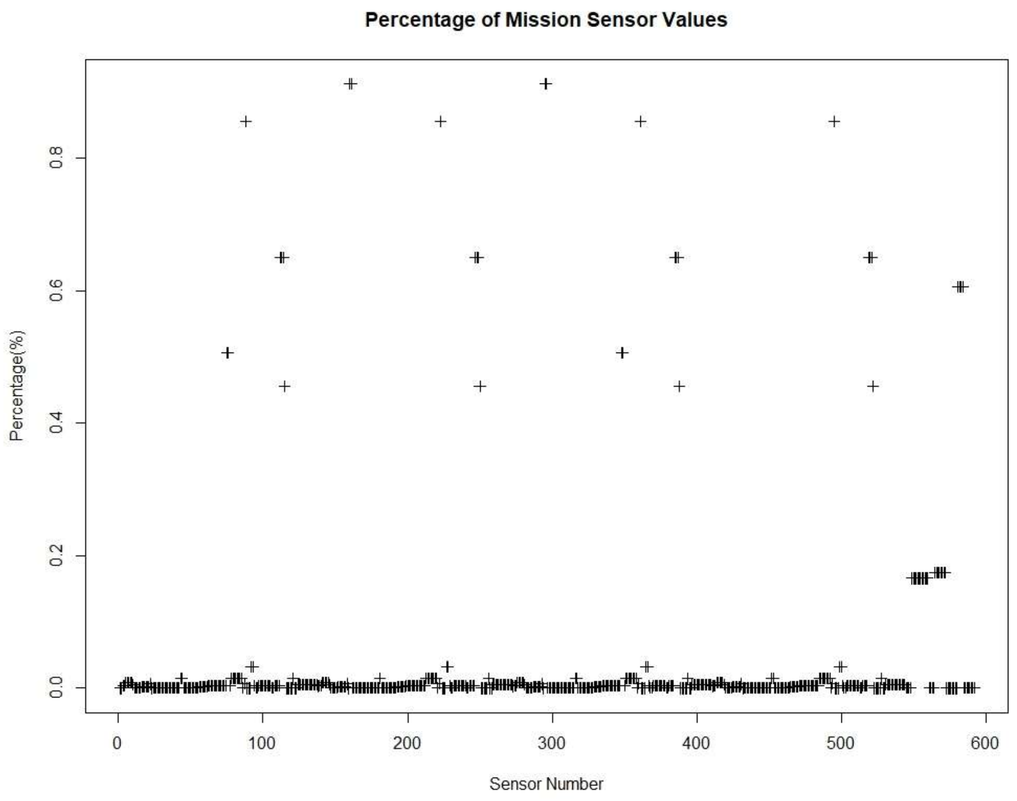
The sensor reading are the features which is a continuous data while the target variables “Class” is a Categorical variable indicating if the respective batch of the semiconductor test is a “Pass/Fail”.

The categorical variables have to be encoded into a numerical variable for modelling purpose.

We can also see that the scale of sensor readings is different. This can result in few of the Sensor reading has a greater influence on the final model. Hence this data needs to be normalized so that all the data are in the same scale.

## Exploratory Visualization

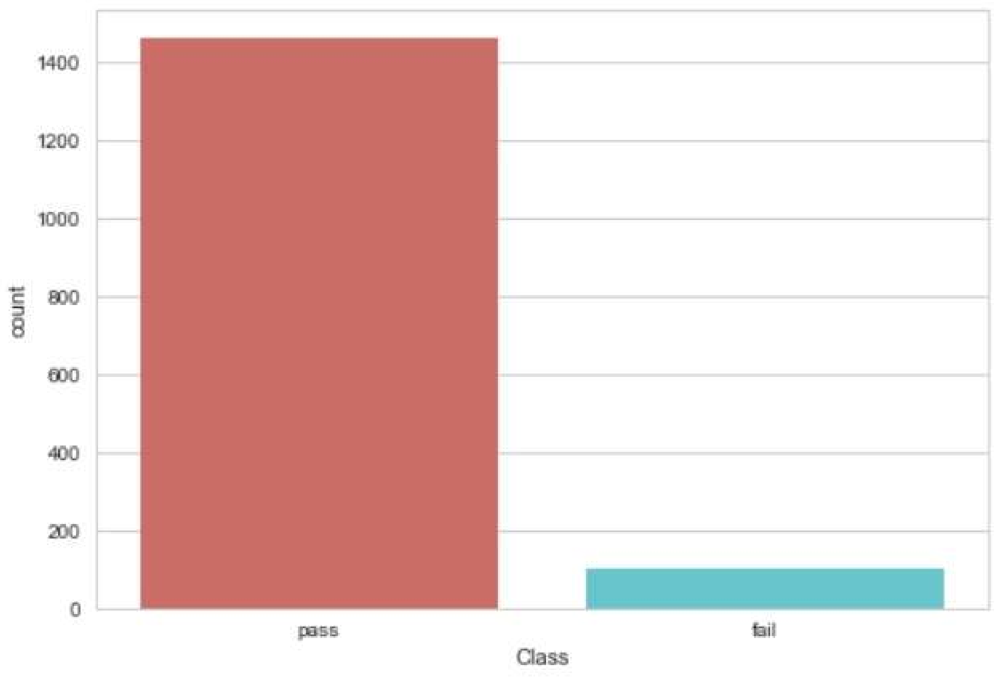
Let’s try to visualize the data to see if there are missing and unique values.



The plot shows the percentage of missing value from each sensor. From the plot its evident that there are many sensors which has more than 40% of the data missing. These sensors readings cannot be reliably used for our predictive modelling purposes. Hence, we will filter out all those sensors from our dataset.

After removing this three will be few sensors still with missing values. These missing values needs to be imputed.

Similarly, we can also see that sensors have a constant reading for all rows. For example, sensor #6 has a reading of 100 for all rows. Such features will have to be eliminated because they add no value to the prediction since there is no variation in their readings.



The plot above is the distribution of the Pass and Fail class in the data set. The class label in the dataset is the dependent variable. From the plot it's obvious that the distribution of class labels is imbalanced. Hence before training our data we need to account for this balance so that the model performs well with the unseen data.

From the visualization of the data we have identified that there will be some sensor readings that needs to be imputed and there is class imbalance problem which needs to be handled before we can build a predictive model

## Algorithms and Techniques

### Data Imputation using KNN

For filling the missing sensor values we will use the KNN algorithm [5] based imputation technique. KNN is an algorithm that is useful for matching a point with its closest k neighbors in a multi-dimensional space. It can be used for data that are continuous, discrete, ordinal and categorical which makes it particularly useful for dealing with all kind of missing data.

The assumption behind using KNN for missing values is that a point value can be approximated by the values of the points that are closest to it, based on other variables.

### Oversampling using SMOTE.

To treat the class imbalance seen in the Class distribution we can logically do two things.

1. Add more failure cases which is the minority Class. This is called Oversampling

2. Reduce the no of observations in the majority Class. This is called Under sampling.

Give the high dimensionality, large no of features vs the observations under sampling will result in very less amount of observations making it difficult for any algorithm to learn from it. Hence, we will do Oversampling of the minority class. We do this by an algorithm called SMOTE. SMOTE stands for (Synthetic Minority Oversampling Technique).

At a high level, SMOTE creates synthetic observations of the minority class (yield Failure) by: Finding the k-nearest-neighbors for minority class observations (finding similar observations). Randomly choosing one of the k-nearest-neighbors and using it to create a similar, but randomly tweaked, new observation. Here oversampling approach does not replicate minority class but constructs a new minority class data instance via an algorithm. Just duplicating the minority classes could lead the classifier to overfitting.

### Logistic Regression

**Logistic regression** models a relationship between predictor variables and a categorical response variable. Here the categorical response is Yield Pass or Failure. Logistic regression helps to estimate a probability of falling into a certain level of the categorical response given a set of predictors (in our case sensor values).

In logistic regression, we use the *logistic function*,

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}.$$

This function takes a value between 0 and 1 for all values of X. The above function can be rearranged and a log can be taken which transforms the function into a linear form.

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X.$$

The left-hand side is called the *log-odds* or *logit*

### Random Forests

**Random forests** are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, bagging repeatedly ($B$ times) selects a random sample with replacement of the training set and fits trees to these samples:

For $b = 1, ..., B$:

1. Sample, with replacement, $n$ training examples from $X$, $Y$; call these $X_b$, $Y_b$.
2. Train a classification or regression tree $f_b$ on $X_b$, $Y_b$.

After training, predictions for unseen samples $x'$ can be made by averaging the predictions from all the individual regression trees on $x$ or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets

## Benchmark

The benchmark model typically used for a two-class classification problem is Logistic Regression. The metric used to evaluate the Model will be discussed in the next section.

# III. Methodology

## Data Preprocessing

As explained in the previous section, the analysis of the data revealed two issues. First is the need to impute missing value and the second is the need to address class imbalance. The first problem was addressed with the help of KNN based imputation algorithm. The class imbalance was addressed by using the SMOTE algorithm which oversamples the minority class again using the KNN algorithm.

## Implementation

After the processing of the data is completed, the first step is to split the data into training and test data in the ration of 80/20. The next step is to fit an appropriate model that can learn from the training data. The output variable in our problem is to predict accurately if a certain batch of semiconductors passed or failed. Once the model has been fitted on the training data it is subsequently tested in the test data for classification accuracy.

The solution to such a problem is a simple binary classification model. The dataset is split into training and test set. The training data has one output - a single binary variable showing the yield result (i.e. a simple pass or fail) for each example. The output of the model is the probability of failure. So, for a given input the model shows the probability that the yield result will be a failure.

After fitting the classification model to the training data, we then evaluate its performance with the test data. Two classification models are proposed. The first one is a simple Logistic Regression Model which is a benchmark model for all classification problems. The second one is the Ensemble Decision Tree Model such as Random Forest which is expected to give a better performance. We will evaluate both the model with respect to the defined evaluation metric below.

## Refinement

The baseline model Logistic Regression yielded an accuracy of 80%. Upon tenfold cross-validation the accuracy score increased to 86.9%. Though the 86.9% accuracy looks good, but for the problem we are trying to solve accuracy may not be a good measure.

We want to know how well I can specifically classify a “positive” conditions, since they’re more important. In statistics, this is called recall, and it’s the number of correctly predicted “positives” divided by the total number of “positives”. In our case the “positive” is the yield failure.

For correctly predicting a “FAIL” condition we have a Recall of 0.38 and for a “PASS” condition we get a recall of 0.84. The AUC value from the ROC curve is also 0.61 which is only slightly better than a random guess value of 0.5.

To improve upon this, we consider a Random Forest based classifier model. Random forest is an ensemble based technique which grows many classification trees. To classify a new object from an input vector, it puts the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

With the default parameter settings for the Random Forest Model we were able to get a Recall value of 0.12 for the Yield Failure condition and 0.97 for the yield Pass condition. This model does a good job of correctly classifying all the Pass cases while does a very poor job in classifying the yield failure. From a business perspective this is not what we want. Our ability to predict a failure case is more important to achieve higher yield and thereby reducing costs.

To improve the Random Forest based classifier model performance we can do parameter tuning. The following three parameters are important from the model performance perspective [6].

***max_features:***

These are the maximum number of features Random Forest is allowed to try in individual tree. There are multiple options available in Python to assign maximum features.

Here are a few of them

*Auto/None* : This will simply take all the features which make sense in every tree. Here we simply do not put any restrictions on the individual tree.

*sqrt* : This option will take square root of the total number of features in individual run. This option is better and does not overfit the data.

***n_estimators:***

This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees give you better performance (makes predictions stronger and more stable) but makes your code slower. This is a tradeoff we need to do.

***min_sample_leaf:***

Leaf is the end node of a decision tree. A smaller leaf makes the model more prone to capturing noise in train data. Generally, its recommended to have a minimum leaf size of more than 50.

With these hyper parameters we have been able to improve the Recall score of the Random Forest Model as shown in the table below.

## IV. Results

### Model Evaluation and Validation

| Model | Recall | Overall Test Accuracy |
|---|---|---|
| Logistic Regression | 0.34/0.84 | 86.9% |
| Random Forest Model (after parameter tuning) | 0.29/0.90 | 91% |

The Random Forest Model overall has a higher classification accuracy in the test data set. Its Recall value of 0.90 in correctly predicting correctly a "Semiconductor Test Pass" condition is also higher than that of the baseline model. However, it does poorly in predicting correctly a "Semiconductor Test Fail" condition which is the most important one.

In this case the simple Logistic Regression Model does perform well. With a K-Fold cross validation the model accuracy ranges from 78% to 93%

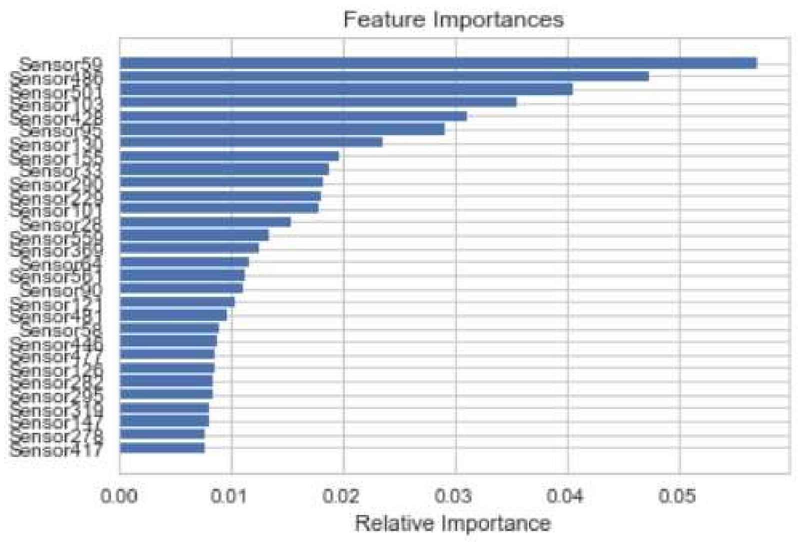
```
[ 0.82978723,  0.79574468,  0.78297872,  0.84255319,  0.8212766 ,
  0.91489362,  0.91452991,  0.93589744,  0.91880342,  0.93162393]
```

Also by varying the Random States we were able to get an average accuracy of 80% consistently.

## V. Conclusion

### Freeform Visualization

The important objective of this project is to understand which of the sensors in the semiconductor manufacturing line is indicative of the wafer successfully passing or failing the test. The plot below shows the relative importance of the sensors in accurately predicting the outcome based on the historical data. This information in turn helps the test engineer in the manufacturing line focus on the stages where a fault is most likely be introduced in the semiconductor manufacturing process.



### Reflection

The most important aspect of this dataset is the highly imbalanced nature of the data with more number of PASS labels (more than 90%) and less number of FAIL labels. This highly imbalanced nature of the dataset makes it difficult for any classification model to learn from the data so that it can accurately predict the correct label for any un seen data.

Hence, we applied a SMOTE oversampling technique for the minority class to address the class imbalance issue before we proceed to fit a classifier model. With the ensemble based decision tree model we were able to get a high classification accuracy of nearly 90% with the unseen data. However, the Recall for “Test Fail” condition is relatively low at 29%.

## Improvement

Between the two models we considered in this design Random Forest only provided a marginal improvement over a simple logistic regression model in terms of accuracy. However, Recall is the most important factor from a business perspective, where logistic regression model performed better. However even in the latter, the maximum recall percentage we could achieve for failure cases is only 34%.

Hence there is scope for investigating other algorithms such as gradient boosting which is less prone to overfitting unlike Random Forests. GBM is a boosting method, which builds on weak classifiers. The idea is to add a classifier at a time, so that the next classifier is trained to improve the already trained ensemble. Notice that for RF each iteration the classifier is trained independently from the rest.

## References

1. https://blog.deepsense.ai/machine-learning-for-applications-in-manufacturing/
2. https://www.kdnuggets.com/2017/10/applied-data-science-solving-predictive-maintenance-business-problem.html
3. http://chem-eng.utoronto.ca/~datamining/dmc/model_evaluation_c.htm
4. https://archive.ics.uci.edu/ml/datasets/SECOM
5. https://towardsdatascience.com/the-use-of-knn-for-missing-values-cf33d935c637
6. https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/