Machine Learning Engineer Nanodegree

Capstone Project Report

Mark Ditsworth

December 2017

Table of Contents

Definition

Project Overview

The global semiconductor market is projected to be worth just under $400B in 2019[1]. The demand for semiconductors driven by digital technology has been relentless since the invention of the integrated circuit. And hot technologies such as graphical processing units, solid-state memory, and digitally controlled power electronics will continue to mandate a large volume of semiconductor production into the future.

As digital tech grows to support, regulate, and maintain our everyday lives, the importance of these devices' reliability also grows. Technologies ranging from implanted heart monitors[2] to inverters connecting utility-scale batteries to the grid[3] are all semiconductor-based technologies that can lead to dangerous outcomes in the event of failure. Thus, it is important that each semiconductor wafer that leaves the fabrication facility for sale must be functional.

However, the sheer volume of these wafers that are produced by individual fabrication prevents process engineers from running validation tests on each semiconductor die and still meeting delivery deadlines[4]. Traditional random sampling runs the risk of missing defective wafers. There is a need for a data-driven solution to detecting a failed semiconductor[5] during the course of production.

Problem Statement

Mass production of semiconductors makes individual testing of each produced IC infeasible, but the reliability of each IC must be guaranteed to prevent failure of the intended application. There must be a method for reliably classifying a device's functionality without

---

[1] https://www.statista.com/statistics/266973/global-semiconductor-sales-since-1988/
[2] https://spectrum.ieee.org/tech-talk/biomedical/devices/medtronic-wants-to-implant-sensors-in-everyone
[3] http://www.afr.com/business/energy/electricity/tesla-battery-responded-to-south-australian-power-failure-in-140-miliseconds-20171220-h08apx
[4] http://anysilicon.com/process-control-high-volume-semiconductor-manufacturing/
[5] B. Pavlyshenko. Machine learning, linear and Bayesian models for logistic regression in failure detection problems. 2016 IEEE International Conference on Big Data. Dec. 2016.

subjecting it to individual electrical and/or thermal testing. A supervised learning algorithm will train a binary classifier to predict the state of a manufactured semiconductor wafer as either 'passed' or 'failed' using available data measured during the manufacturing process. Various statistical analysis techniques will need to be applied to determine the subset of the available data that will serve as useful features to the classifier, and to reduce the dimensionality as much as possible.

Metrics

An effective classifier for predicting the functionality of a semiconductor device should have an accuracy above 90%. Accuracy is defined in terms of confusion matrix components by the equation below.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \times 100\%$$

Accuracy is included as one of the metrics due to the ubiquity of the term. If this model were to be proposed to the owner(s) of the semiconductor manufacturing plant who are less versed in machine learning theory, they would relate best to a high accuracy figure.

Furthermore, to ensure proper functionality despite the skewedness of the data set, the Matthews Correlation Coefficient should be above 85%. The Matthews Correlation Coefficient is a measure of the quality of binary classifications, which works well on differently sized classes[6]. It is defined in terms of confusion matrix components by the equation below.

$$MCC = \frac{(TP \times TN) + (FP + FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(FN + FN)}} \times 100\%$$

It should be noted that in the event the equation above results in an undefined value, the Matthews Correlation Coefficient is defined as 0.

[6] http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf

Analysis

Data Exploration

The data set comes from the University of California at Irvine Machine Learning Repository, posted by Michael McCann and Adrian Johnston[7]. It consists of 1567 examples of silicon wafer measurements from 590 sensors in a semiconductor manufacturing plant, with each wafer's pass or fail label. What each of these sensors are measuring is not given, thus any knowledge of the semiconductor fabrication process will not aid in the selection of features. The table below shows the statistics for a select few sensors.

Inspecting the *Count* statistic – the number of examples for a corresponding feature that are not missing – shows that missing data is prevalent in the data set, especially for sensors like 581, which has over half of its data missing.

Sensor	**0**	**5**	**9**	**581**	**586**
Count	1561	1553	1565	618	1566
Mean	3014.452896	100	-0.000841	97.934373	0.021458
Std	73.621787	0.0	0.015116	87.520966	0.012358
Min	2743.24	100.0	-0.0534	0.0	-0.0169
Median	3011.49	100.0	-0.0013	72.2889	0.0205
Max	3356.35	100.0	0.0749	747.3048	0.1028

Table 1: Selected Data Set Statistics

By looking at *Mean* and *Std*, the variance of the data can be analyzed. Sensor **5** has a standard deviation of 0, thus the sensor value remains constant throughout the data set. It must be assumed that other sensors could have 0 standard deviations as well. For the other sensors, the standard deviation must be analyzed relative to the mean to determine how much the sensor measurement changes throughout the examples. Sensor 581 has a standard deviation close to the mean values, while sensor 0 has a standard deviation two orders of magnitude less than the mean. Compared to 581, sensor 0 is much more stable throughout the measurement examples and the differences in the values may be due to measurement noise.

Of the total 1567 measurement examples, only 104 examples are for semiconductor wafers labeled as 'failed'. Thus, the data set is heavily skewed towards the 'passed' wafers.

[7] http://archive.ics.uci.edu/ml/datasets/SECOM

Comparing the correlation coefficients between each sensor allows the redundancy of the data to be estimated. While 590 sensors are available, if several sensors effectively measure the same quantity, the dimensionality of the features given to the classifier can easily be reduced. The heat map in Figure 1 illustrates the correlation coefficients. Aside from the diagonal line showing a 1.0 correlation coefficient between each feature and itself, there are several other areas of high correlation: two more diagonal lines and six pockets. Thus, it can be expected that removal of redundant features will significantly reduce the dimensionality of the problem.

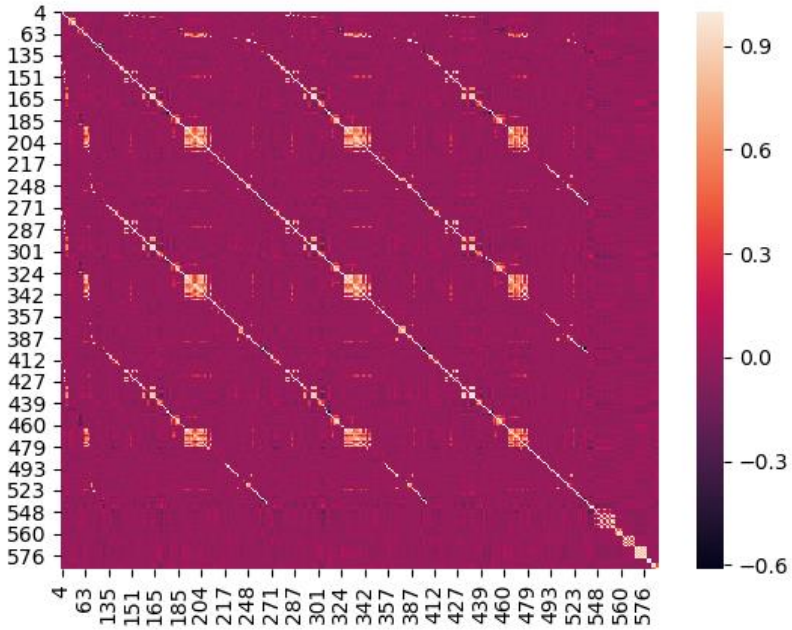

Figure 1: Heat map of correlation coefficients

Techniques and Algorithms

Preprocessing

Before the data can be used to train a binary classifier, it must be pre-processed to be more useful. First, missing data must be handled. The easiest method of dealing with missing data is to ignore examples with missing features. This has a disadvantage of decreasing the data set size, and potentially missing out on valuable information.

Replacing missing data in an example with the average value of the sensor across all examples allows the example to be used in training the model. But there must be a much larger number of existing examples for the sensor in order for the average value to be meaningful. If there is too much missing data for a sensor, the inclusion of said sensor as a feature will likely do more harm in the model than good. Thus, features that are too sparse should be ignored.

In addition to removing features that are too sparse, features that are too constant (add no variance to the data set), should be removed as well. Comparing features' variance first requires feature scaling. One common scaling method in SK-Learn is `StandardScaler`. This method scales each feature by transforming it to a distribution with a unit standard deviation. This makes the comparison of feature variance impossible. Another scaling method in SK-Learn is `MinMaxScaler`, which scales each feature such that they are between two fixed numbers, usually [0,1] or [-1,1].

Once the features are scaled, the variances of the features can be compared. A common metric for evaluating variance is the coefficient of variation (CV), which is defined as the standard deviation normalized by the mean[8]. A CV much less than one signifies that the data hardly varies from the mean, while a CV much larger than one signifies that the data set contains a wide variety of values around the mean. Features with CVs less than one should be considered for removal, as their variances are more likely to be a result of measurement noise, and may not aid in classification. It should be noted that the CV is only defined on positive data sets.

Skewed Data

As mentioned in the Data Exploration section, the data set is skewed towards functioning semiconductors by about 14:1. There are several methods commonly used to handle skewed data sets: undersampling of the majority class, increasing the minority class weight, and synthetic minority oversampling technique (SMOTE)[9].

Undersampling of the majority class simply consists of selecting a subset of the majority class examples such that the size of the two classes are more comparable. This clearly eliminates the skewedness of the data set; however, it decreases the size of the data set which could lead to problems with bias or variance in the classification model.

Using class weights allows the complete data set to be used, but places more emphasis on the minority class examples by weighting them more in the model.

---

[8] Everitt, Brian (1998). *The Cambridge Dictionary of Statistics*. Cambridge, UK New York: Cambridge University Press.

[9] Tom Fawcett. 'Learning from Imbalanced Classes". Silicon Valley Data Science. August 25. 2016. https://www.svds.com/learning-imbalanced-classes/

SMOTE is a method of generating more examples for the minority class to reduce the skewness of the data set. For each data point in the minority class, *k* nearest neighbors are selected. The midpoints between the root data point and each of the nearest neighbors are added to the data set for the minority class. The number of nearest neighbors is selected based on how much growth is desired for the minority class. The class size will grow by $k \times 100\%$. Like undersampling, SMOTE will un-skew the data set. But by generating synthetic data to mimic authentic data, it makes the assumption that if more data were available for the minority class, it would resemble its current distribution.

Principal Component Analysis

Principal Component Analysis (PCA) is a method of changing the basis of the data set to new orthogonal components to capture, such that the variance captured by each component is maximized[10]. After the orthogonal components are found and the data set is transformed to the new basis, the first *n* components that capture a sufficient percentage of the total data set's variance can be used as features for a classifier, and the remaining components can be ignored. In this way, the dimensionality of the problem seen by the classifier is reduced without needing to eliminate any of available features.

Logistic Regression

Logistic Regression is a linear classifier. It is most useful when there is a near-linear separation between the two classes of data, but through tuning of the regularization parameter, it can tolerate certain degree of non-linearity and still achieve high fidelity.

Support Vector Machine

Support Vector Machines (SVM) fit a hyperplane that best separates two classes of data. This is similar to Logistic Regression, except that SVM commonly employ a 'kernel trick' to classify data that is non-linearly seperable. The kernel function operates on each data point to add additional dimensionality, thereby allowing non-linearly separable classes to be separated by a hyperplane, as shown in Figure 2[11].

[10] http://scikit-learn.org/stable/modules/decomposition.html#pca
[11] https://prateekvjoshi.com/2012/08/24/support-vector-machines/

Decision Tree

Decision tree classifiers create a tree of yes/no decision nodes based on the features in the data set. The order of feature examination is chosen based on the amount of information gain/entropy loss of each feature. Decision trees work well on highly non-linear data sets, but are prone to over-fitting if preventative steps are not taken, such as limiting the depth of the tree.

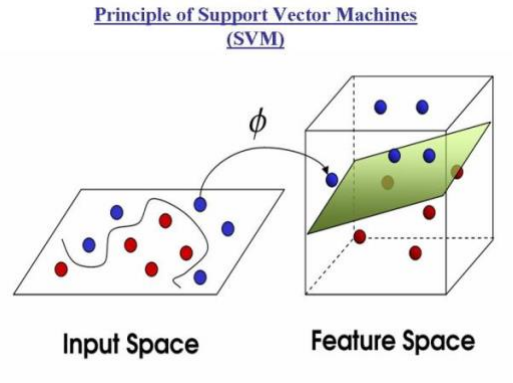

Figure 2: SVM kernel trick example

AdaBoost Ensemble Learner

AdaBoost uses *n* number of 'weak learners', which are typically small decision trees. Individually, the accuracy of each learner is only a little better than randomly guessing. The output of the collective learner is determined by a weighted majority vote of the weak learners. When training, a single learner is trained first. The examples that the first learner incorrectly learns are then weighted higher for the successive weak learner, so that it prioritizes the classification of the missed examples[12].

Benchmark

The benchmark model will be a binary classifier using logistic regression that uses a number of sensors' data as features, and predicts the state of the semiconductor wafer: functional or non-functional. The benchmark model will be compared with more advanced classification models (SVM, decision tree, and AdaBoost ensemble learners) based on their MCC and accuracy metrics.

[12] http://scikit-learn.org/stable/modules/ensemble.html#adaboost

Methodology

Data Preprocessing

To begin, the data set and corresponding labels were imported as pandas DataFrames. The `describe()` function performed the necessary statistical calculations for each sensor (feature). Features with missing data could be identified by their count being less than 1567. Features with more than 20% of their examples missing were removed from the data set because, as detailed earlier, they would not provide meaningful information for the classifiers. This process resulted in a dimensionality reduction of 32. After removal of these empty features, the missing data in the remaining 558 features were replaced with their average values using `fillna(df.Data.mean())`.

The next step in preprocessing the data is to remove features that are constant, or who's variation is likely due to measurement noise. The metric used to determine viability is the Coefficient of Variation (CV). As detailed earlier, this first requires feature scaling. Since the standard deviation of each feature will be compared to the others, and the CV is only defined on positive data sets, `MinMaxScaler` is used to rescale each feature to within [0,1]. After scaling the data set, any features with a CV below 0.3 are deemed too constant, and are removed from the data set. A CV threshold of 0.3 is selected because due to the distortion of the feature distribution by `MinMaxScaler`, a 0.3 CV corresponds to a much lower CV of the pre-transformed feature (assuming the positive value requirement is held). There were 201 features determined to be too constant, reducing the number of features to 357.

After removing empty features and non-varying features, redundant features are eliminated. The `corr()` function in pandas serves to calculate the correlation coefficient of each pair of features. The heat map above in Figure 1 illustrates the correlations between each feature. It can be seen that aside from the main diagonal where the correlation coefficient would be expected to be 1, there are several other areas with high magnitudes of correlation. Thus, it is expected that many features will be eliminated due to redundancy.

Feature pairs with correlation coefficients higher than 0.9 were selected for removal. The features to drop were identified from these selected feature pairs using the pseudo-code

below. The result of this routine ensured that a set of *n* highly correlated features is reduced to just a single feature from the set.

```
for pair in selected_pairs:
        if first feature < second feature:
                add second feature to drop_list
drop_list = unique values in drop_list
```

Figure 3: Pseudo-code for removal of redundant features

From the remaining data set, 183 features were identified as redundant, thereby reducing the number of features to 174.

Handling Skewed Data

At this point, the data set has been cleaned of empty, constant, and redundant features, leaving 1576 examples of 174 features belonging to one of two classes: 'passed' and 'failed'. The problem posed by the 'passed' class outnumbering the 'failed' class by 14:1 still exists. Four different techniques were implemented to alleviate this problem: undersampling, class weights, and synthetic oversampling (SMOTE). This section will describe the implementation of these techniques. The results of each technique on the performance of the model will be discussed in the results section.

Undersampling is simply selecting a subset of the majority class comparable in size to the minority class to serve as the data set for the majority class. This was implemented by separating the data set by class, and randomly selecting 104 rows (examples) from the majority-class data set. This new data set was then concatenated with the minority class data set to create the undersampled data set equally composed of 'passed' and 'failed' examples.

Class weights can be assigned to make the model more sensitive to a certain class when minimizing the cost function during training. Since, the 'passed':'failed' ratio of the data set is about 14:1, the weight of the 'failed' class is set to 14, with the 'passed' weight kept at 1. Class weights are set in Sk-Learn when initializing the model with a dictionary {*class label* : *weight*} as such:

```
clf = LogisticRegression(random_state=0, class_weight={1:14})
```

Synthetic Minority Oversampling TEchnique (SMOTE) generates more minority class examples by finding the *k*-nearest neighbors to minority-class data points, and adding the midpoints to the data set as minority class examples. SMOTE was implemented by using the imbalanced-learn module for Python[13]. There are three variants of SMOTE available in imbalanced-learn: borderline-1, borderline-2, and svm. All versions neglect 'noisy' data points where all their nearest neighbors belong to an opposite class. Borderline-1 and 2 find nearest neighbors from 'in-danger' points, where at least half of the nearest neighbors belong to the opposite class as the selected point. Borderline1, unlike borderline2, adds midpoints only between selected points and the nearest neighbors that are in the opposite class. Svm uses SVM classifiers to find boundaries for each class, and limit the generation of minority-class examples to its decision region.

Principal Component Analysis

After preprocessing was completed, and skewedness was addressed, principal component analysis (PCA) was performed to further reduce the number of features required by the classification model. PCA was performed with the Sk-Learn `PCA().fit()` function. Figure 3 below shows the composition of the first five principal components and their respective explained variances. The cumulative sum of the explained-variance ratios for each principal component were

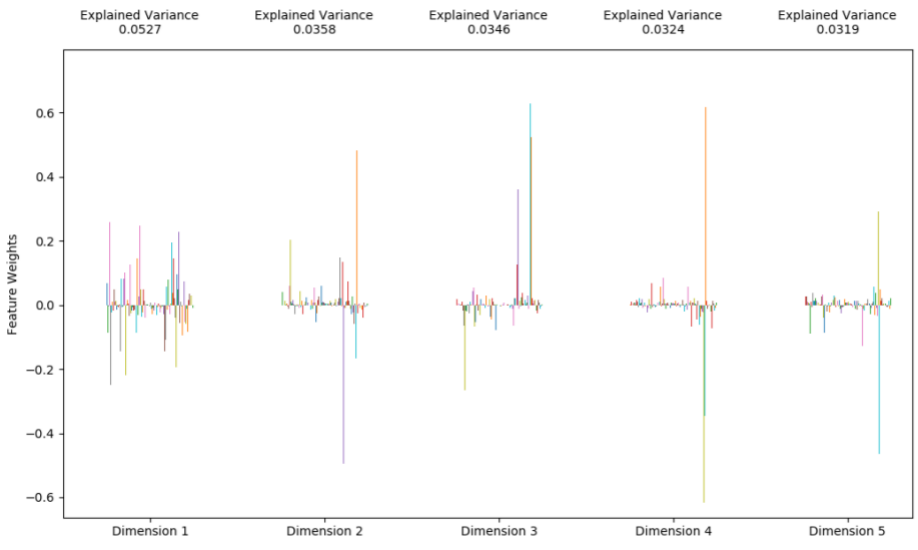

Figure 2: First 5 principle components

[13] http://contrib.scikit-learn.org/imbalanced-learn/stable/index.html

analyzed to determine that 70 principal components were required to capture over 90% of the variance in the data set. 90% was selected partially due to the intuitive understanding that 90% of the variance should be sufficient, and partially due to the graph of explained variance ratio vs number of principal components shown in Figure 4. The elbow point is near the midpoint between 0.8 and 1, hence the selection of 90%. This allows the 174 features to be decrease by 60% through the PCA transformation, reducing the computational load on the model.

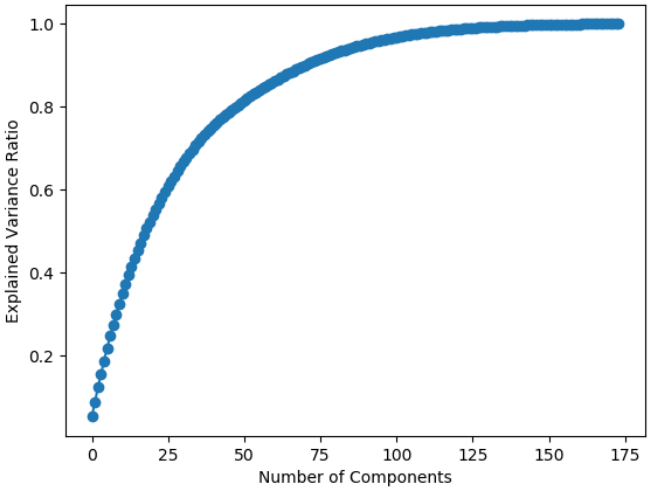

Figure 4: Explained variance ratio vs num. of principal components

Classifier Implementation

All classifiers were implemented using SK-Learn. Their parameters were optimized using GridSearchCV to exhaustively search through a range of values, with the Matthews Correlation Coefficient as the scoring method. The table below enumerates the parameters swept by GridSearchCV.

Parameter	**Log. Regress.**	**SVM**	**Decision Tree**	**Adaboost**
Regularization (C)	[0.01,2]	[0.01, 2]		
Kernel		rbf, poly(3-5)		
Max Depth			[10,100]	
Num. of Learners				[10,50]
Learning Rate				[0.1,5]

Table 2: Values of each parameter sweep for each model

Results

The best results of each classifier (out of five iterations with different random_states) for each skewed data approach are explained below.

Undersampling

	Logistic Regression	**SVM**	**Decision Tree**	**AdaBoost**
Accuracy	63.5%	63.5%	38.5%	51.9%
MCC	0.286	0.273	-0.233	0.026

Undersampling the 'passed' class resulted in poor performance for the each of the models. Logistic Regression performed the best with a MCC of 0.286, far below the target metric of 0.85.

Class Weights

	Logistic Regression	**SVM**	**Decision Tree**	**AdaBoost**
Accuracy	67.9%	46.4%	77.0%	
MCC	0.079	0.054	-0.029	

Using class weights to mitigate the skewed data problem resulted in very poor performance of each model. While the accuracy gets as high as 77% with a decision tree, the MCC of each model is close to 0, signifying that the models are performing no better than randomized guessing. The relatively high accuracy of logistic regression and decision tree likely only results from the random selection of the training and test set.

It should be noted that the AdaBoost classifier model in SK-Learn does not allow for the assignment of class weight, and thus the Accuracy and MCC were not recorded for comparison in this skewed data approach.

SMOTE

	Logistic Regression	**SVM**	**Decision Tree**	**AdaBoost**
Accuracy	82.9%	62.9%	88.4%	89.7%
MCC	0.674	0.676	0.773	0.758

Applying the SMOTE algorithm performed the best of all the skewed data techniques. The Decision tree model scored the best with a MCC of 0.773 and AdaBoost scored the best in accuracy with 89.7% (with only a slightly lower MCC than Decision Tree). These scores place the model below the specified metrics for success.

Robustness

With a small data set, quantifying the robustness of the model is best done by retraining each model with different random states and comparing the mean-to-variance ratios of the MCC. The higher the mean:variance ratio, the more robust the model is due to the smaller variance relative to the average value. The table below shows the robustness quantity for each skewed data approach. The sample size for each skewed data approach is five.

	Logistic Regression	**SVM**	**Decision Tree**	**AdaBoost**
Undersampling	30.4	7.7	1.8	11.1
Class Weight	53.9	41.5	13.9	
SMOTE	1245.3	4050.1	404.6	3905.1

Using SMOTE led not only to the best MCC and accuracy in the models, as seen in the previous section. The technique also created models that are more robust. Of the four models implemented SVM had the highest robustness metric, but not much higher than AdaBoost. However, considering the superior performance of the AdaBoost in terms of MCC, it would be best to sacrifice a small amount of robustness for a much better performing model.

Justification

The benchmark model, logistic regression, failed to meet the performance metric specifications of 90% accuracy and 0.85 MCC. The best outcome was only 82.9% accurate with an MCC of 0.674. The advanced models scored much better, but also failed to meet the

performance standard. The best classifier would be the AdaBoost classifier. It's top performance was comparable to the Decision Tree classifier, which ranked best in terms of MCC. But the robustness of the AdaBoost was nearly an order of magnitude higher than that of the decision tree. And while the AdaBoost classifier failed to meet the performance metrics, the differences are small; improvements could be made to attain the accuracy and MCC specifications, as will be described in the next section.

Conclusion

Reflection

This project allowed for a deeper dive into data processing techniques, which I believe are the core of machine learning engineering. People smarter than I have done the work to develop the algorithms we have today, and implement them into plug-and-play formats like SK-Learn. The difficult part is that real-world data is typically far from ideal and must be cleaned and polished before anything meaningful can be extracted from it.

This project began with data pre-processing: removing useless features based on quantifiable metrics, replacing missing data with mean values, performing PCA for dimensionality reduction, and handling the skewedness of the data set through three different techniques (undersampling, class weights, and SMOTE). For each skewed data approach, the benchmark logistic regression model was trained, optimized, and evaluated against three more advanced models: SVM, decision tree, and AdaBoost.

Free-Form Visualization

None of the models were able to meet the performance standards, but while there are improvements that could be made in the approaches taken, the most significant factor is the data itself. As seen in Figure 5, there is significant overlap between classes in some of the features. Overlap like this will most certainly lead to misclassifications. The best way to improve the functionality of the models in this project (or in any project for that matter), is to improve the relevance and fidelity of the data set.

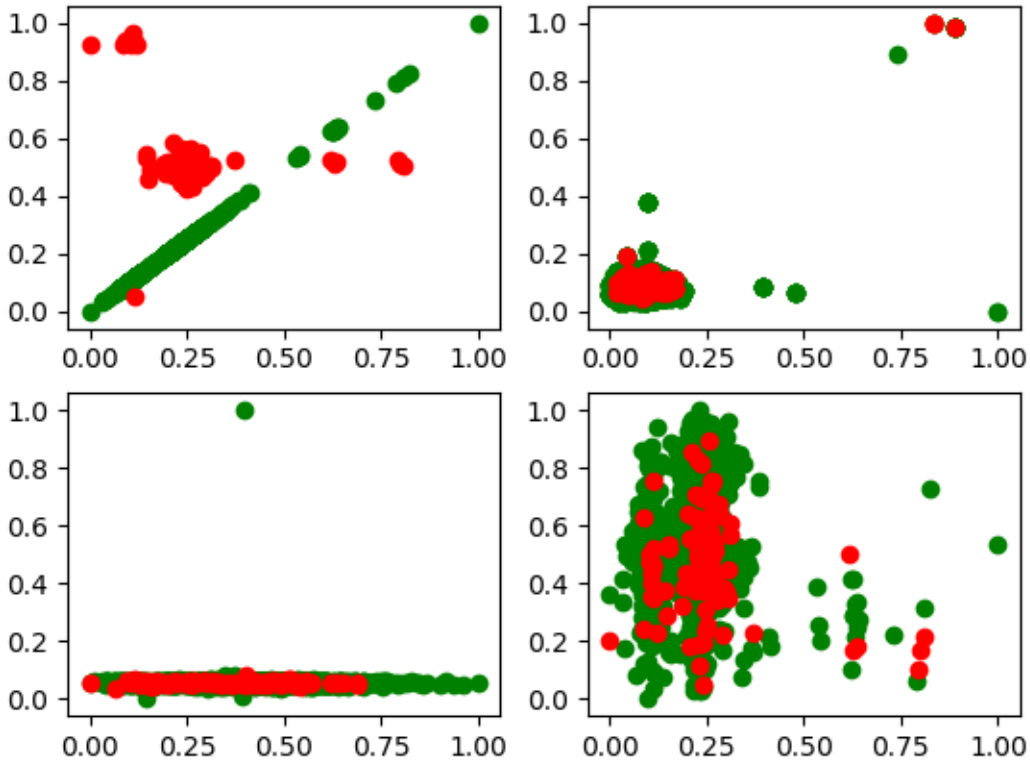

Figure 5: Four examples of two different features plotted against each other. 'Passed' class is green; 'failed' class is red. It can be seen that while some features are clearly separable from others in terms of class label, other pairs of features are indistinguishable

Improvement

As the models did not quite meet the benchmark metrics of 0.85 MCC and 90% accuracy, further improvement would be needed to be successful. The crux of the problem appears to be that the data is too noisy/irrelevant to the classification of semiconductor wafer functionality. If the sensors available in the data set were labeled, an engineer familiar with the manufacturing process could identify the most relevant features, and by working with the manufacturer, additional required sensors could be placed to aid in the classification capability.

But assuming these changes cannot be made and the current data set is all that is available, improvements could still be made. Firstly, the thresholds used in preprocessing (emptiness, CV, correlation coefficient) can be refined. Secondly, the required percentage of variance needed to be captured by the PCA components could be raised from 90% to 95% or even 99%. This will increase the number of features used to train the models, thereby

increasing the computational complexity. However, it could refine the capabilities of the classifier.

The method of handling missing data could also be improved. In this project, missing values were replaced with averages of the respective feature. Instead, supervised learning could be used to train regressors for each feature, and the missing values replaced with the predicted value based on the other features. Assuming decent regressors for most features can be attained, this may increase the fidelity of the data set and yield a better performing classifier.

Lastly, Tomek link removal[8] applied on the training set will clarify the boundary between the two classes. This may allow the model to find a better decision boundary by reducing the noise where the two classes intermingle. However, it is also possible that the clearer boundary created by Tomek link removal will result in a model that suffers from high bias, and will poorly classify the semiconductor wafers.