1 Introduction

Semiconductor manufacturing processes are highly technologically complicated processes with over a thousand steps, inevitably being prone to error due to how delicate and repetitive the stages they have to undergo are. Normally wafers are built out of different layers tangled with circuits and, if the final product is defective, it represents waste for the companies in terms of not only money but also time.

The need for a practical and automated solution to detect failures in early stages is evident, not only to avoid material waste but mainly to avoid releasing faulty products to the market. Therefore, in this work statistical analysis and modelling algorithms were used to develop such a solution, testing many different approaches and methods to define the ideal option for maximizing the quality of the prediction model.

2 Business understanding

Business objective

Lower down cases of defective wafers so as to increase productivity and reliability by means of prediction using data provided by sensors installed on semiconductor process.

Business success criteria

Reducing faulty semiconductors during manufacturing process to get 100% Quality Semiconductor products.

Data goals

1. Come up with a feature selection technique that chooses from over the 500 features, which are provided by sensors scattered across different semiconductor fabrication, the ones with the most potential so as reduce considerably the model’s processing time;
2. Since Data is imbalanced due to the amount of pass and fail cases, it is necessary to look for a technique that could help to compensate and help the model to increase accuracy;
3. Develop a model to detect on-time any possible fault raising during the wafer production.

3 Data understanding

Data description

Based on the descriptions in Kaggle and Research Gate, the SECOM dataset, provided by Michael McCann and Adrian Johnston, contain real data on the semi-conductor manufacturing process. The data comes from 590 sensors and process measurement points spread through the whole production line, in this case consisting of 1567 entries with the sensor data as well as a timestamp, ID and a pass/fail label. On the image below it is possible to visualize the distribution of pass and fails through the dataset.

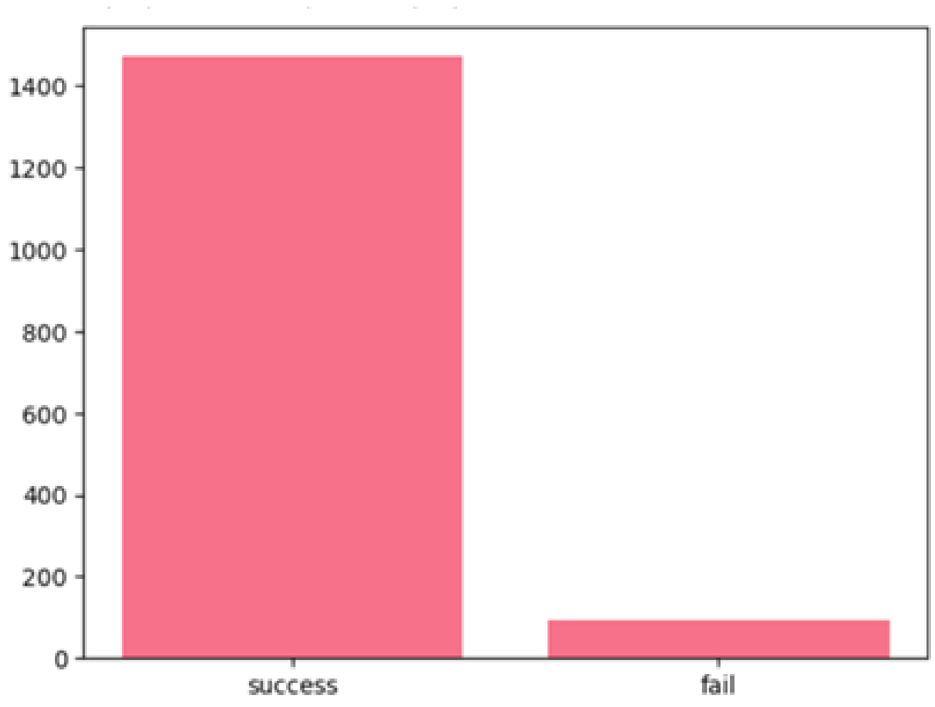

Fig.1: Majority and Minority Classes

All data was in *float 64* format except the timestamp.

4 Initial data preparation

The first steps, after loading the data from the .sav file, where as follows:

1. Drop the *ID* and *timestamp* columns, as they are not used for modelling;
2. The *class* column is the categorical value (pass/fail) which is used to building the model, so it is defined as the *target*;
3. Split the data into Train and Test datasets using the *train_test_split* from *sklearn*, dividing the features as the **X** dataset and the *class* as the **Y** dataset, while segregating 80% of data for training and 20% for testing.

5 Feature removal

The feature removal techniques used consist of the following techniques:

- Excluding features containing more than 55% missing values.
- Excluding features containing values with variance below 5%.

For the first step it was used the python library *missingno*, which helped visualize the distribution pattern of null values across the dataset.

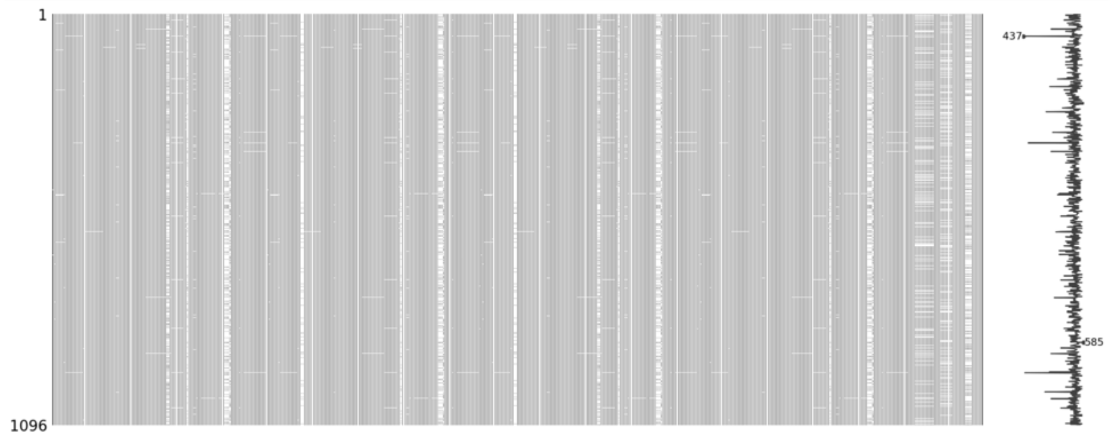

Fig.2: Null value distribution using *missingno*

This technique reduced dimension of dataset from 588 features to 580 features. After that the low variance values were handled, utilizing the following formula to define the features to be dropped:

$$\mathbf{Var[X] = p(1\text{-}p) = 0.95*(1 - 0.95)}$$

After this process the dataset is optimized from 580 features to 464 features.

6 Outliers

The next step is dealing with the outliers in the dataset. For that, two methods were used:

1. Using the inter-quartile range (IQR), which is the distance between the first and third quartiles of a dataset. More specifically, the IQR tells us the range of the middle half of the data (Singh et al, 2006). Values outside the 1,5 times the Inter-quartile range are converted to *NaN*.
2. Afterwards, based in the z-score which is calculated based on the deviation from the mean in relation to the standard deviation. (Cousineau et al, 2010) This value is used to remove outliers based on z score greater than 3 times the standard deviation and replacing them with *NaN*.

7 Imputation

In this step were also used two methods:

KNN (or K-nearest neighbor algorithm)

The idea in the *kNN* method is to identify *k* samples in the dataset that are similar or close in the space to the missing value. We can then use these samples to estimate the value of those missing data points. Each sample's missing values are imputed using the mean value of those k-neighbours in the dataset (Pan et al, 2015.)
KNN was used for both test and train datasets using the KNN imputer available in the *sklearn* library.

MICE (or Multivariate imputation by chained equations)

A very sophisticated approach, **MICE** involves defining a model to predict each missing feature as a function of all other features, repeating this process of estimating feature values multiple times. (Hayati et al, 2015)
In other words, it creates a copy of the main dataset and fits the mean value for the missing data and then runs a regression model, keeping one value as null and other 2 values with mean value, repeating this until a best fit value is found.

8 Feature Selection

Although in many of the trial models one feature selection function was used, due to the fact that the models themselves have methods to select features (evident as this feature selection modules use models inside themselves), the best results were actually found when skipping this step. Nevertheless, two methods were used:

RFE

This method is a wrapper-type feature selection algorithm which also internally uses filter-based selection. This means that a machine learning algorithm (in our case Linear Regression) is used as the core of the method and “wrapped” by RFE (Chen, 2007), being used to rank features.
This method searches for feature subsets by fitting the given machine learning algorithm used in the core of the model, ranking features by importance, discarding the least important features. By starting with all features in the training dataset and repeating the steps we can reduce the number of features to select a defined number of them.

Boruta

The algorithm uses a wrapper approach built around a random forest (Breiman, 2001) classifier (Boruta is a god of the forest in the Slavic mythology). It is relatively quick, usually running without parameter tuning, and gives a numerical estimate of the feature importance. It is an ensemble method in which classification is performed by voting of multiple unbiased weak classifiers | decision trees. These trees are independently developed on different bagging samples of the training set. The importance measure of an attribute is obtained as the loss of accuracy of classification caused by the random permutation of attribute values between objects. It is computed separately for all trees in the forest which use a given attribute for classification. The accuracy loss average and standard deviation are then computed.

9 Balancing

Data imbalance is one of the main characteristics of this dataset due to its unequal distribution of pass and fail entries. Among the 1,567 entries there are only 95 fail cases while the remaining 1,472 being pass cases, the ratio of majority-to-minority being 15:1.

To deal with this imbalance we used the Synthetic Minority Over-sampling Technique (**SMOTE**), boosting the number of samples in the minority by creating random entries calculated between two other random minority entries. In Python, we have the *imblearn* library that offers a solid **SMOTE** implementation, as well as other resampling techniques. (Chawla et al,2002)

10 Modelling

For each iteration of our test models, different steps of the data preparation methods were used. In general, all methods above were applied to both the Train and Test datasets, except in the Imputation step (were only one of the methods was used) and the feature selection step (were either one or none method was used). After the SMOTE technique was applied, the dataset was ready for training the model.

Cross-validation to train and test the models

Cross validation comes into the picture due to the need of defining better parameters, this is a powerful technique and the idea behind it is quite simple: it enables taking the train dataset and split it into K different subsets also known as K-folds, then iterating K times, training the model with different parameter subsets. The test subset will be always different as well as the combination of subsets to train the model. (Munirathinam et al, 2006)

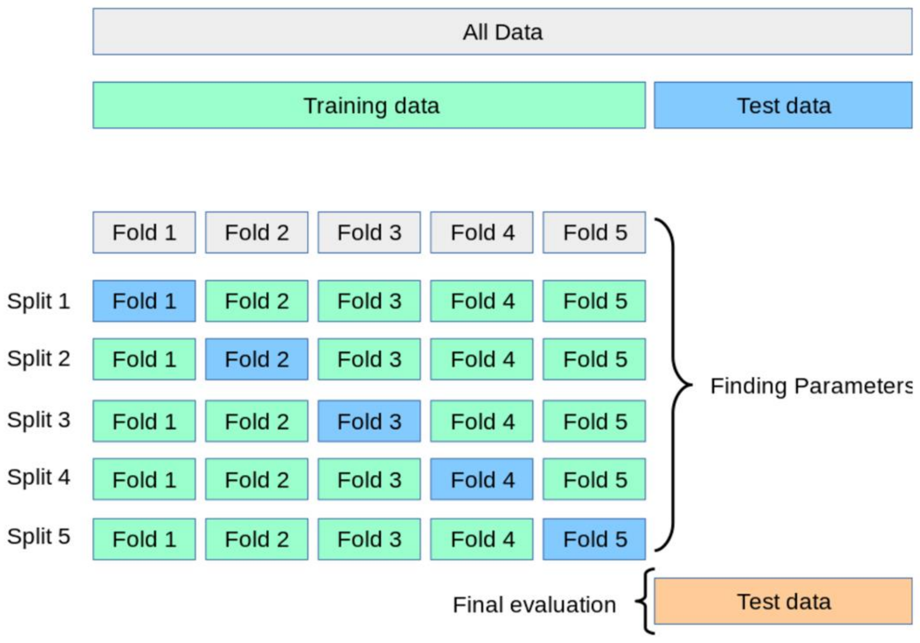

Fig.3: Cross-validation example

Since at every iteration the results vary when training the model with different subsets of train and test datasets, these are averaged and used to evaluate the model as a whole.

Models implemented

Decision Tree Classifier:

This algorithm takes a set of inputs and maps the various outputs that are a result from the consequences or decisions. To reach a decision (classification) it is based on a result of various hierarchical steps. (Munirathinam et al, 2006)

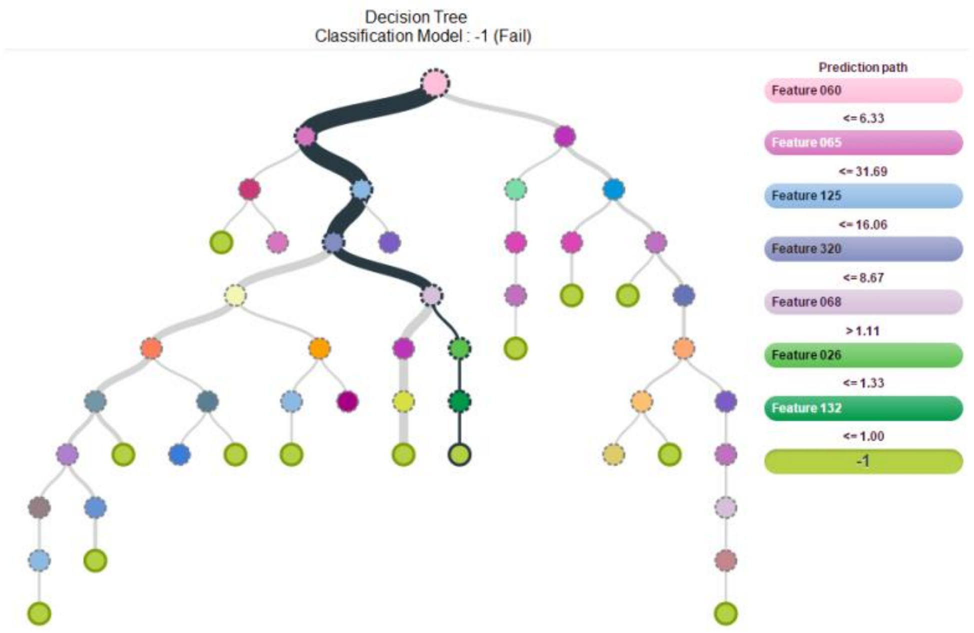

Fig.4: Decision tree example

Random Forest:

Based on the Decision trees classifier, the model builds a "forest" that is an ensemble of decision trees trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result. In other words, random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. (Breiman , 2001)

Naive Bayes:

This is based on the Bayes theorem. The classification is based on the highest probability considered. It does this by predicting the membership probabilities for each class such as the probability that given record or data point belongs to a particular class. (Munirathinam et al, 2006)

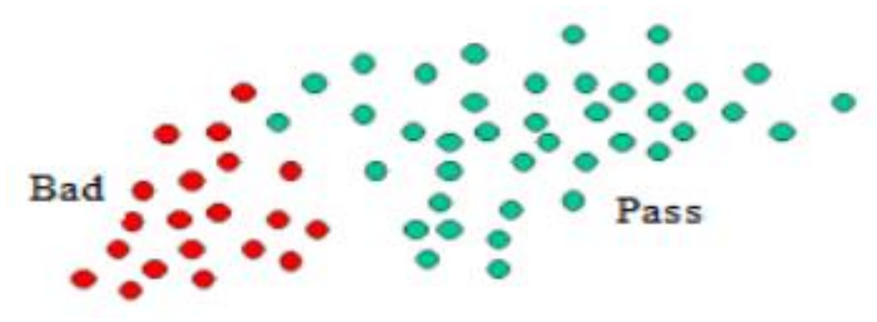

Fig.5: Naïve Bayes example

Logistic Regression:

Finding the best fit line to create a binary classification and identifying the area where how far or in which area of the division of the line the observation lies or and hence can be predicted to. Here the prediction for the output is transformed using a non-linear function called the logistic function. (Munirathinam et al, 2006)

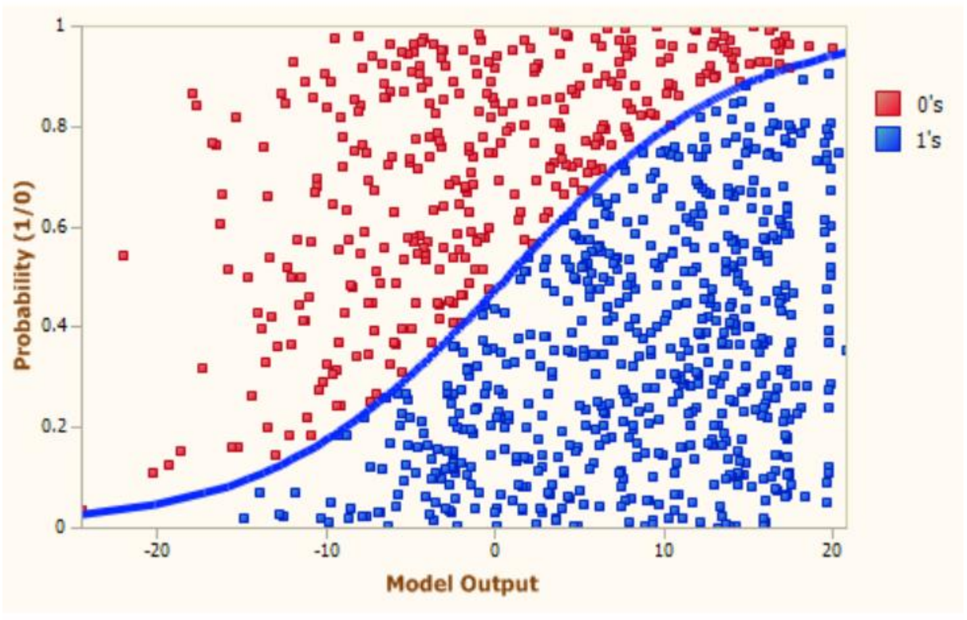

Fig.6: Logistic regression example

Support Vector Machine:

The plotting is carried out in the n-dimensional space. Each feature's value is the value of the specific coordinate. And we find the ideal hyperplane which differentiates between the two classes. So, these support vectors represent the coordinate representations of each individual observation. (Munirathinam et al, 2006)

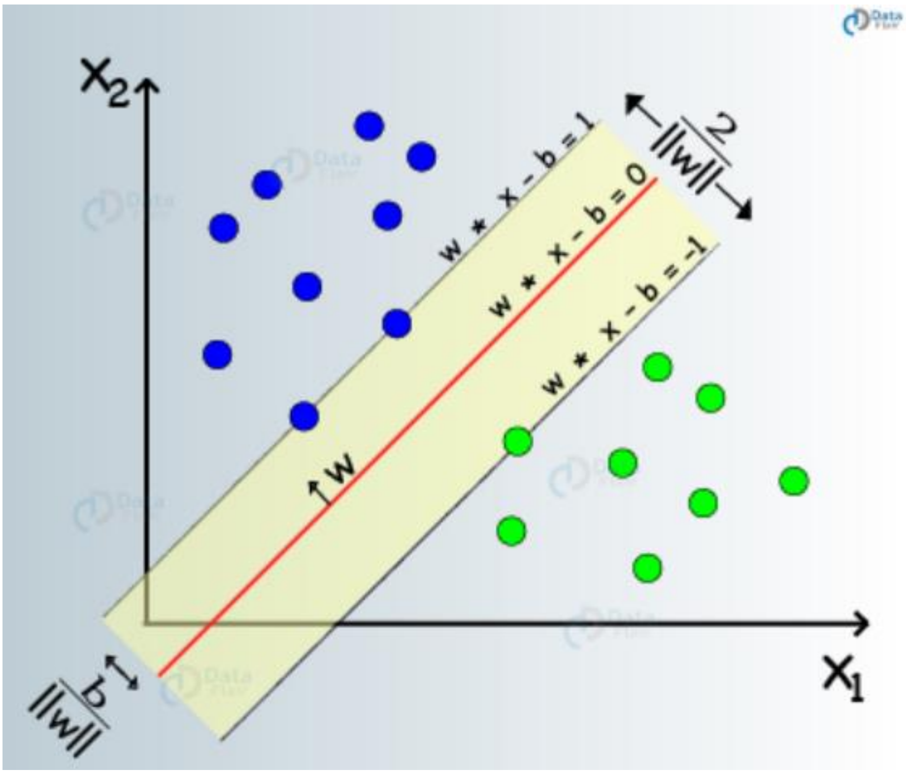

Fig.7: SVM example

XGBoost

XGBoost stands for Extreme Gradient Boosting. It works based on the gradient boosting decision tree algorithm. This is also an ensemble technique where the previous model errors are resolved in the new models. Then these new models are added straight away until a new improvement is seen. (Chen et al, 2016)

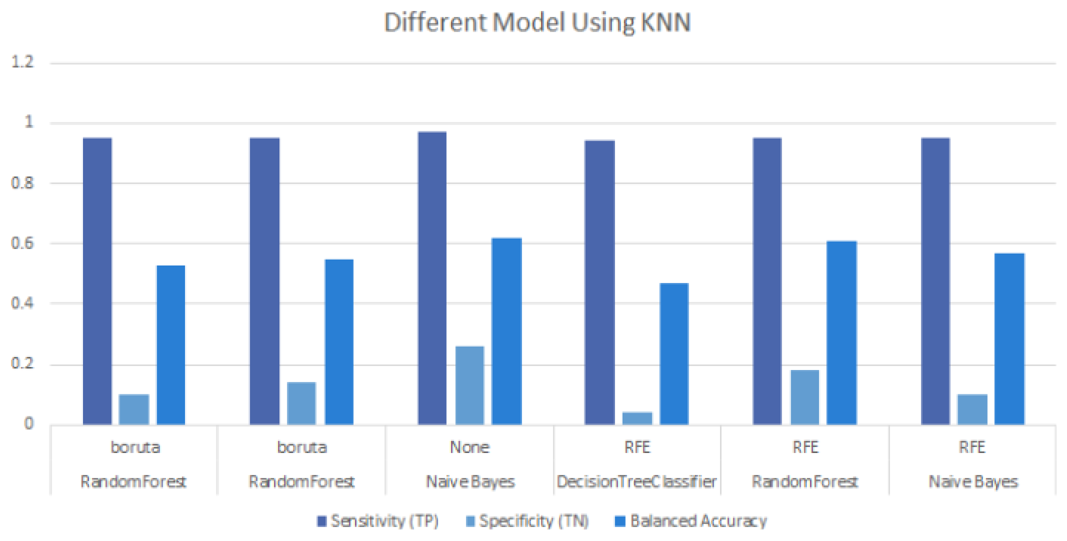

Fig.8: Results from different models

11 Evaluation

As per the CRISP DM model this is the 5th phase of the Data Mining where the models built will be evaluated for the performance as an important measure to take decision about which model is best suitable for deployment for the specific businss case. **Daniel T. Larose (2005) states that “It is of critical importance that these models be evaluated for quality and effectiveness before they are deployed for use in the field” (p.200). Also, we need to find the right evaluation techniques as different cases requires different performance metrics (Alice Zheng).**

In SECOM classification case we experimented with 22 combinations from different techniques of Feature Selection, outlier removal, imputation, with and without data scaling, with and without Grid search, and classification algorithms. Since the prediction model is a classification model, the following classification evaluation metrics is used from the results obtained from 22 variations of models on the test data set.

The famous evaluation metrics used for the classification performance are the Accuracy, confusion matrix, log loss, ROC, AUC, precision-recall. The ones used in this experimentation are follows:

1. Confusion Matrix (True Positives, True Negatives, False Positives, False Negatives)
2. Specificity
3. Sensitivity
4. Type1, Type2 errors

VALIDATION

Alice Zheng (2015) states that “The model must be evaluated on a dataset that is statistically independent from the one it was trained on. Why? Because its performance on the training set is an overly optimistic estimate of its true performance on new data.”

CONFUSION MATRIX:

Confusion matrix is a squared matrix which provides the results in a squared matrix of True Positives (TP), True Negatives (TN), False Positives (FP) and False negatives (FN). Hence for choosing the model we used Confusion matrix evaluation method.

Ian H. Witten Eibe Frank Mark A. Hal explains that:

"The cost of misidentifying problems with a machine that turns out to be free of faults is less than the cost of overlooking problems with one that is about to fail. The true positives (TP) and true negatives (TN) are correct classifications. A false positive (FP) is when the outcome is incor[1]rectly predicted as yes (or positive) when it is actually no (negative). A false negative (FN) is when the outcome is incorrectly predicted as negative when it is actually positive…Good results correspond to large numbers down the main diagonal and small, ideally zero, off-diagonal elements."

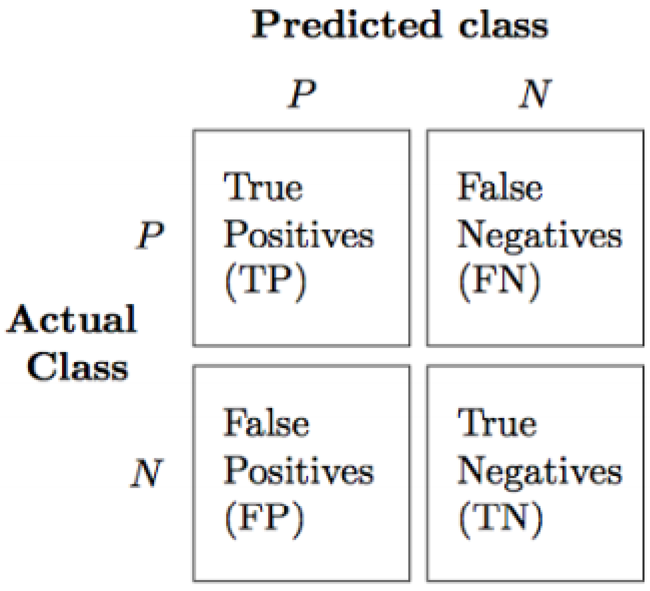

Fig.9: Definition of classes in the confusion matrix

Specificity

Specificity is also referred as True Positive Rate (TPR), or Recall. It is a measure of the correctly positive predicted to the total number of positive samples. (Tharwat, Alaa (2021))

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{P}$$

Fig.10: Specificity formula

Sensitivity:

Also referred to as True Negative Rate (TNR), or inverse recall. It is a measure of the correctly classified negative samples to the total number of negative samples**.** (Tharwat, Alaa (2021))

$$TNR = \frac{TN}{FP + TN} = \frac{TN}{N}$$

Fig.11: Sensitivity formula

The specificity and sensitivity are great measurement criteria especially when imbalanced dataset exists.

Type1 and Type2 Errors:

Type 1 Error (FN): Positive sample getting classified as Negative
Type 2 Error (FP): Negative sample getting classified as Positive

Final selected model: Naive Bayes

After evaluating the performance of the models involved in the train and test phase, Naive Bayes was the model chosen that better predicted the defective products.
This algorithm belongs to the supervised and classifiers models type, Gaussian Naïve Bayes would look into the probability of the two classes (Good and defective products) depending on certain features and observations, by

leveraging a Gaussian distribution and standard deviation and determining its final prediction based on an “score” in other words the one with the highest probability.

This type of algorithm can handle large amounts of features, also they are really fast at processing such amounts of data. However, what makes them so fast is that contains few tunable hyperparameter. For instance, the function provided on python to implement Naïve Bayes is called GaussianNB only receives two parameters prior probability and var smoothing, the former allows to modify directly the probabilities for the target feature instead of being calculated from the dataset and the latter helps to include more observations that are far away from the mean by smoothing the curve.

Although the algorithm only has two parameters, hyperparameter optimalization was carried about aiming to improve the performance of the model by looking at the best values for both prior probability and var smoothing. This was achieved by using a python function provided by the library Sckit Learn called GridSearchCV, this is a powerful function since not only allows to look for the best hyperparameters but also do cross validation at the same time.

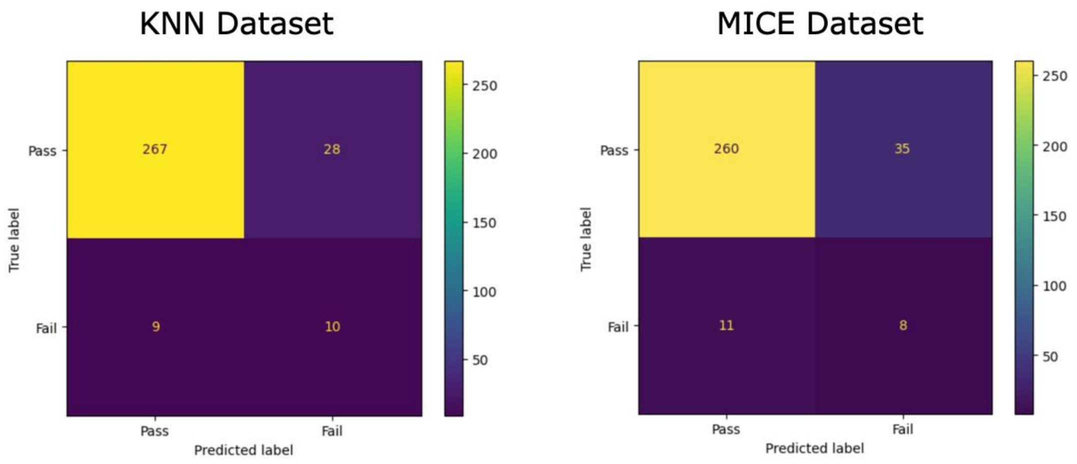

Fig.12: Confusion matrix without Hyperparameter optimization

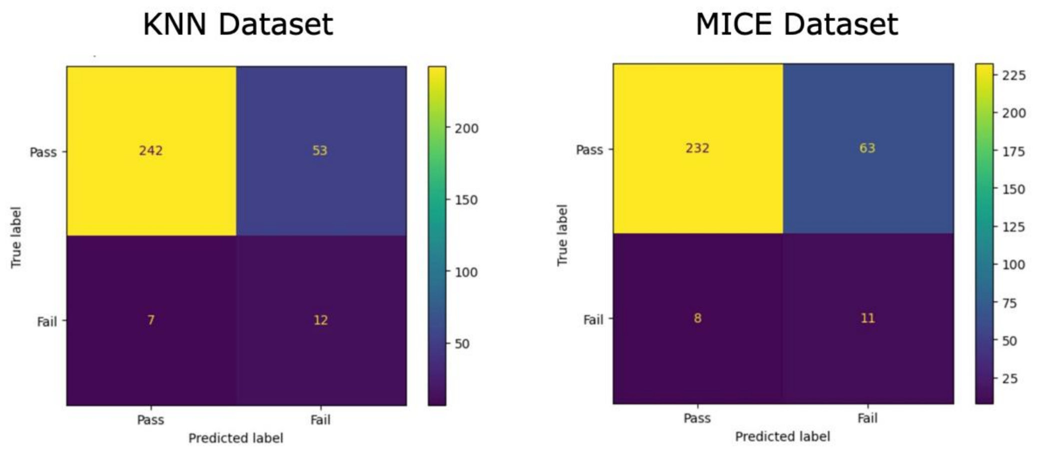

Fig.13: Confusion matrix with Hyperparameter optimization

Since naive bayes gave us good scores compared to other model, having test 314 samples in 2 different scenarios, i.e., with and without hyper parameter.

→ Without hyper parameter and with KNN , TP was 267 and FP was 28 out of 295 pass and coming to fails of 19 samples FN was 9 and TN was 10.

→ Without hyper parameter and with MICE , TP was 260 and FP was 35 out of 295 pass and coming to fails of 19 samples FN was 11 and TN was 8.

→ With hyper parameter and with KNN , TP was 242 and FP was 53 out of 295 pass and coming to fails of 19 samples FN was 7 and TN was 12.

→ With hyper parameter and with MICE , TP was 232 and FP was 63 out of 295 pass and coming to fails of 19 samples FN was 8 and TN was 11.

Based on this value the model performs better with KNN imputation and also without using hyper parameters.

One of the disadvantages of this algorithm is that it assumes that all features are independent, but one can assume that for example the quality of health care depends on individual's income or that the weight of an object growth with its size. In other words, assuming that all incoming features are independent is definitely a Naïve way to look into the problem, that is why the name of this model. Therefore, it is important to take care of dependent features making use of Feature selection techniques, such as Pearson correlation.

12 Conclusions

This project was conducted to define the best model building process to identify faulty wafers in a semiconductor production line based on the feedback from the numerous sensors in such line. After a series of experiments following the CRISP DM process, we identified that the Naive Bayes model built from a subset of features selected by a gain ratio criterion can detect the fault cases at the very high rate of 97%. But the detection of true negative is only 26% which is not sufficient to detect damaged chips. with average accuracy of 88.2% and Balanced accuracy of 62% we need to improve the boosting technique or change the hyper parameter to improve the Fail detection rate.

In the end, although we have defined the best model based on our various test runs, it is clear that the final model can still be refined to achieve even better results. Considering the various possible variations, we have only partially checked all possibilities for the most effective model, yet we have gained many insights as to the workings of the models and could certainly, with more time, define a more precise and trustier model.