Machine Learning

Defect Prediction on Production Line

Team:
AIT BACHIR Romuald
ALLEMAND Fabien
FLORET Arthur
JARDOT Charles

Abstract

Contents

List of Figures

1 Introduction

The goal of this project is to develop an AI based solution for a real world industry problem faced by Valeo [1].

Valeo [2] is an industry leading French automotive supplier. In order to stay competitive, the company wants to develop a system that is able to identify defects on products before testing.

Four files containing relevant data are available. The data are mainly values measured during production on different mounting stations as well as additional measures performed on test benches.

The target is to find the best prediction: Output = f(inputs)

2 Data Analysis

The goal of this section is to take a first look a the data in order to understand what it represents, then proceed to an in-depth analysis of said data.

2.1 Data Description

Data is contained in four different comma-separated values (csv) files: inputs and output for both training and testing. The very first line of each file correspond to the headers. The following lines contain informations about a given product where each product is identified by a unique code:

ID = **PROC_TRACEINFO** = It's a unique code given to the product.
Example: I-B-XA1207672-190701-00494.

- XA1207672 is the reference.
- 190701 is the date: here 01st of July of year 2019.
- 00494 is the unique code given to the product, whatever it happens, the product will have this id number frozen forever.

This number is increased by 1 each time we process a new product, every 12 seconds. So for example: I-B-XA1207672-190701-00495 is the next product.

Input features are measures (numerical values) collected on different assembly stations with the sensors or devices connected to Programmable Logic Controllers which are storing all of them to keep the full quality traceability. Examples : OP070_V_1_angle_value, OP120_Rodage_I_value...

This is the result value of OP130 (test bench). Value 0 is assigned to OK samples (passed) and value 1 is assigned to KO samples (failed). This is the combined result of multiple electrical, acoustic and vibro-acoustic tests.

2.2 In-Depth Analysis

Let us put the testing data aside for now and take a closer look a the training data.

The training data consists of 13 measures (inputs features) made during production line on just shy of 35000 products. This measures correpond to various forces, angles, screwing torques... By taking a look at the distribution for each feature (Figure 1) we can clearly see that every one of them has a different shape.

A quick glance at the pairwise correlation matrix reveals that there are no obvious correlation between any features (Figure 2).

Looking at the results of a PCA (Figure 3 4) or at multiple histograms (with items grouped by class for each feature, as displayed in Figure 5), there is no feature that clearly helps separating the two classes.

An in-depth look at the box-plots (Figure 6) reveals that there might be some outliers in the dataset.

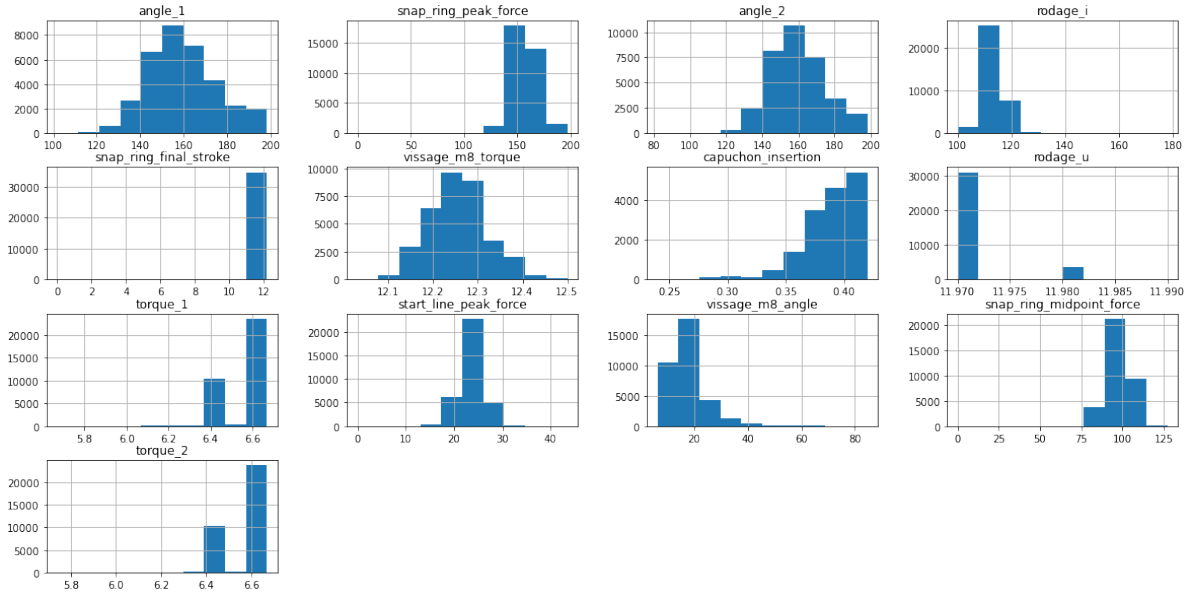


Figure 1: Distributions of the input features

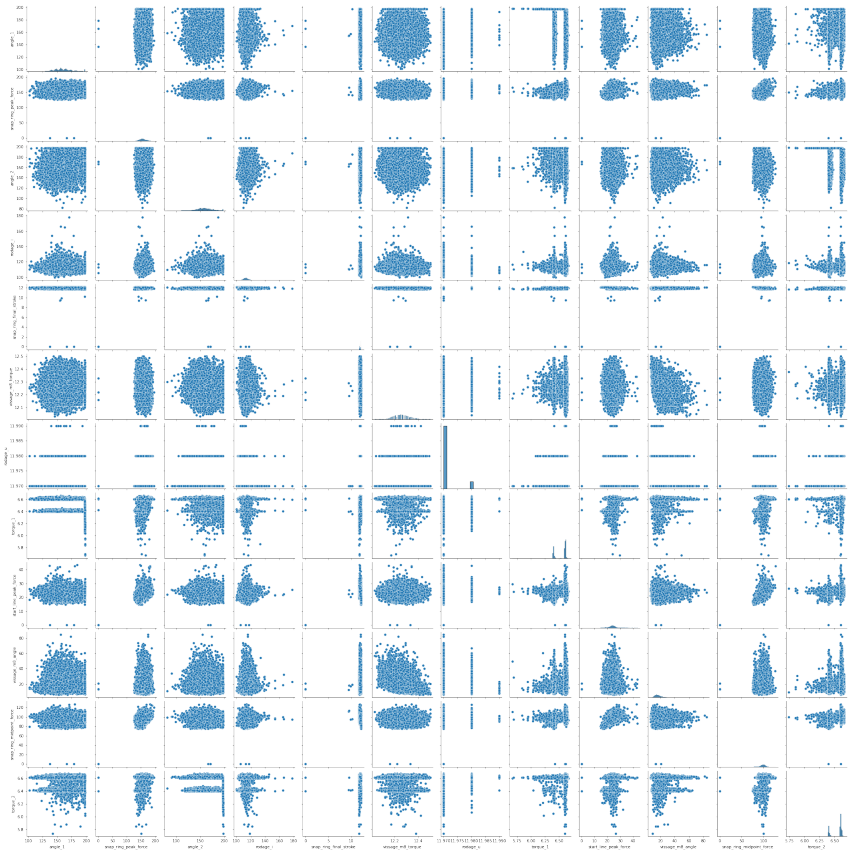


Figure 2: Features pairwise correlation matrix

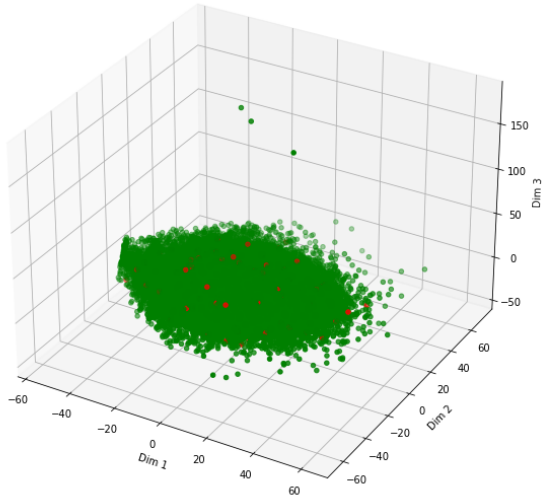


Figure 3: PCA results (projection of the training population on the three most relevant dimensions)

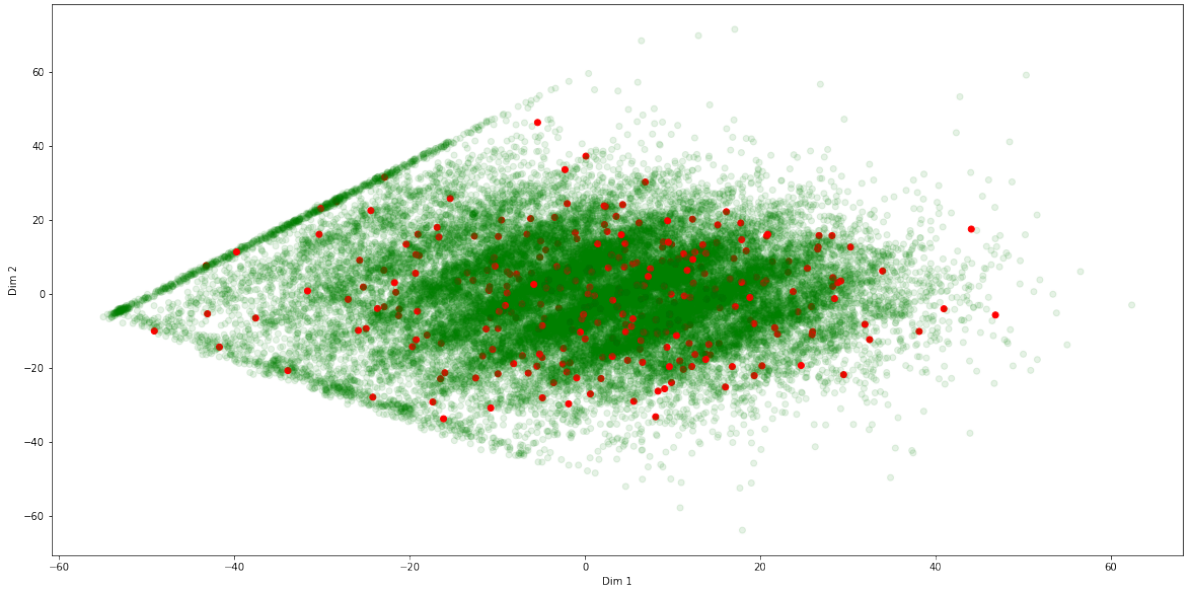


Figure 4: PCA results (projection of the training population on the two most relevant dimensions)

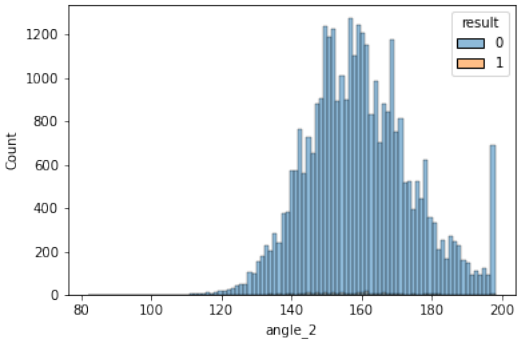


Figure 5: Distribution of the *angle_2* feature with items grouped by class

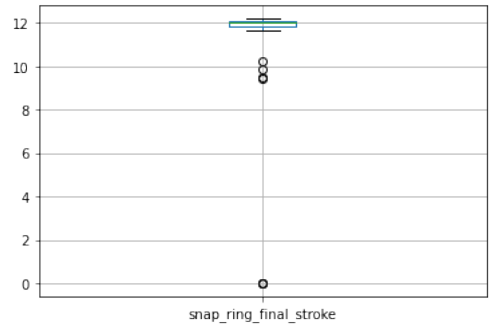


Figure 6: Box-plot of the *snap_ring_final_stroke* feature

One feature requires more attention. *capuchon_insertion* is the only feature containing missing values. In fact, more than 50% of the sample population are missing this value.

The training data also contains the result value of the test bench for each product tested on the production line. As expected, there are only a few of defective items. Although this is good for the company, it means that **less than 1%** of the training population are defective items. The effectives of the two classes are very unbalanced which could really impact our models.

3 Data Preparation

The data analysis performed in the previous section revealed some flaws in the dataset. The two most noticeable are the presence of outliers and the population imbalance between the two classes. These two flaws are known for harming the learning of most ML models. Different attemps at solving these issues are presented in this section.

3.1 Outliers

3.2 Balance Classes

The following paragraphs deal with class balancing, explaining the importance of balanced classes and three different approaches two rectify the imbalance.

As said in the previous section, a mere 0.008% of the population is defective individuals. In other words, the class of individuals which needs to be detected is truly under-represented in the dataset. Without any pre-processing such a dataset is very unlikely to give good results with most of the ML models. As

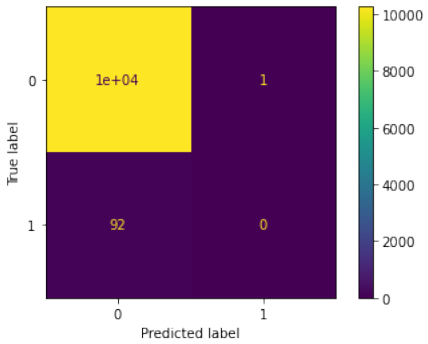


Figure 7: Confusion matrix of a k-Nearest Neighbors classifier with the base dataset

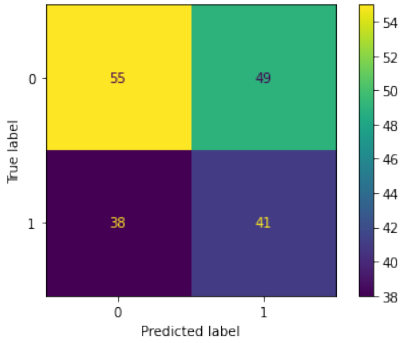


Figure 8: Confusion matrix of a k-Nearest Neighbors classifier with the dataset balanced by removing individuals

shown in Figure 7, there are not enough defective individuals for the classifier to properly learn how to distinguish a defective individual from a valid individual. As a result, the classifier simply over-fit the valid class.

Remarque | Over-fitting the over-represented class is very hazardous as the accuracy of the classifier will remain really high eventhough the classifier is not classifying anything. Hopefully the confusion matrix shows the classification details.

The easiest method to deal with unbalanced classes is to remove valid individuals from the training dataset. This process is fairly straight forward (valid individuals can be randomly selected in the dataset) and does not require a lot of processing (selected individuals are simply deleted).
Figure 8 clearly shows that the classifier is not over-fitting the valid class anymore. In that sense, the results are better but they are still far from what is expected from a performant classifier.

This method is very arguable for two reasons. First, deleting individuals obviously implies a loss of data. Eventhough the individuals are randomly selected, some important features can disappear. In other words, the remaining data is not necessarily representative of the global population. On top of that, the resulting dataset is smaller meaning the model will have less instances to learn on. In this problem, this will dramatically impact the results because there is already few data.

Of course it is also possible to work the other way around and artificially create individuals instead of deleting others. The easiest method is actually to duplicate defective items until there are as many as the valid ones. This compensate the imbalance between the two classes without knowing anything about the features values. Figure 9 shows the effect on the classifier is significant.
Another way that has been implemented during this project is to slightly modify individuals during the duplication. This way new individuals are actually new individuals (not only duplicates) adding information to the overall dataset. This method assumes individuals from the same class are close to one another on

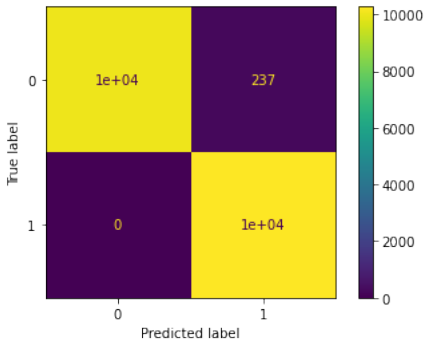


Figure 9: Confusion matrix of a k-Nearest Neighbors classifier with the dataset balanced by duplicating individuals

every dimension (i.e. regarding every feature) and changing a feature's value by a small amount will not induce a change of class. The way it has been implemented is surely not ideal but rather intuitive. A random amount of features are selected during each "duplication" and the value is modified by a very small percentage (typically 0.005).

The final method in order to balance classes consists in leaving the classes inbalanced in the dataset but during computation attributing a more significant weight to the individuals from the under-represented class. Technically speaking this is equivalent to duplicating individuals but it requires less memory, it is easier for computation and is already implemented for some algorithms from well known ML libraries.

4 Model Selection

Choosing the ML model that works the best for a given problem is a key factor for good results but it can be very time consuming. The following pages present the results obtained with several ML algorithms. Each method will be briefly introduced and tested using the training dataset. The resulting classifier are evaluated using three different tools:

- Confusion matrix
- Precision, recall and F1 score
- ROC AUC Score

These three tools are presented and explained in Section 4.1.
Knowing the data is labeled all the methods but the last one are based on supervised learning.
Moreover it has already been demonstrated in Section 3.2 that balancing classes dramatically improves results so the ML models are tested with an augmented training dataset containing duplicated individuals.

4.1 Evaluation Method

There are many tools to analyse the performances of ML models. Three tools will be thoroughly used later. The goal of the following paragraphs is to give a quick understanding at what they represent and how to use them properly.

The confusion matrix is a tool specifically used for classification problems. This matrix simply represents the results of the classification made by a given classifier. For instance in Figure 9, the classifier classified 237 valid items as defective (top left).
By looking at a confusion matrix, it is fairly easy to see what the classifier did wrong by comparing it to the ideal matrix which would contains zeros for the false positive (top right) and false negative (bottom left)

emplacement.

The values contained in the confusion matrix also allow to compute the precision, recall and f1 score of a classifier:

- Precision: how accurate is the classifier when classifying a defective item (closer to one is better)
- Recall: how good is the classifier at finding defective items in the population (closer to one is better)
- f1 score: the harmonic mean of precision and recall (closer to one is better)

Precison and recall are linked values meaning there is always a tradeoff: if precison rises, recall tends to decrease. So it is important to adjust precision and recall according to the application. In this problem, a company would most likely prefer to double check an item rather than selling a defective one so a strong recall would be ideal, at the expence of precision.

Finaly, Receiver Operating Characteristic (ROC) curve plots the true positive rate against the false positve rate (i.e. sensitivity against specificity, Figure 10). It allows to distinguish the classifier from a random classifier. It is generally associated to the ROC Area Under the Curve (AUC) score. A ROC AUC score over 0.5 means the classifier is better than a random classifier. An ideal classifier would show a ROC curve following the top left edge of the graph and a ROC AUC score of one.

4.2 Naive Bayes Classifier

Naive Bayes classifiers use Bayes'theorem making the "naive" assumption that every pair of features are independant given the value of the class variable [4] (i.e. $P(x_i|y, x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i|y)$) leading to the following equality:

$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)} = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..., x_n)}$$

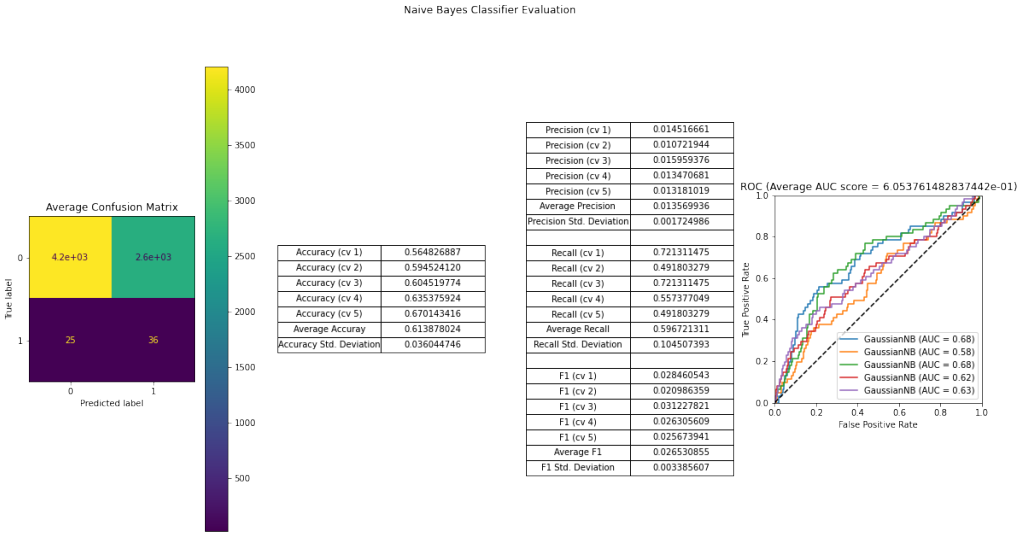
The following results correspond to a Gaussian Naive Bayes [7] classifier that is very common when dealing with numerical continuous values. However it also assumes that every features follows a Gaussian distribution.

The confusion matrix in Figure 10 shows poor results. This is expected because Figure **??** demonstrates some features do not follow a Gaussian distribution. In addition, the assumption that features are independant from one another might not be relevant with this dataset. For instance, the *snap_ring_peak_force* might increase with the value of *anlge_1*.

4.3 k-Nearest Neighbors

The k-nearest neighbors algorithm works by classifiying an individual by looking at its neighbors [0]. The classifier simply look at the k nearest neighbors of the individual in the training population and attributes the class that correspond the best to the neighbors (the neighbors vote for their class, each vote is weighted either evenly accros the neighbors or according to the distance between the individual and the voting neighbor [8]).

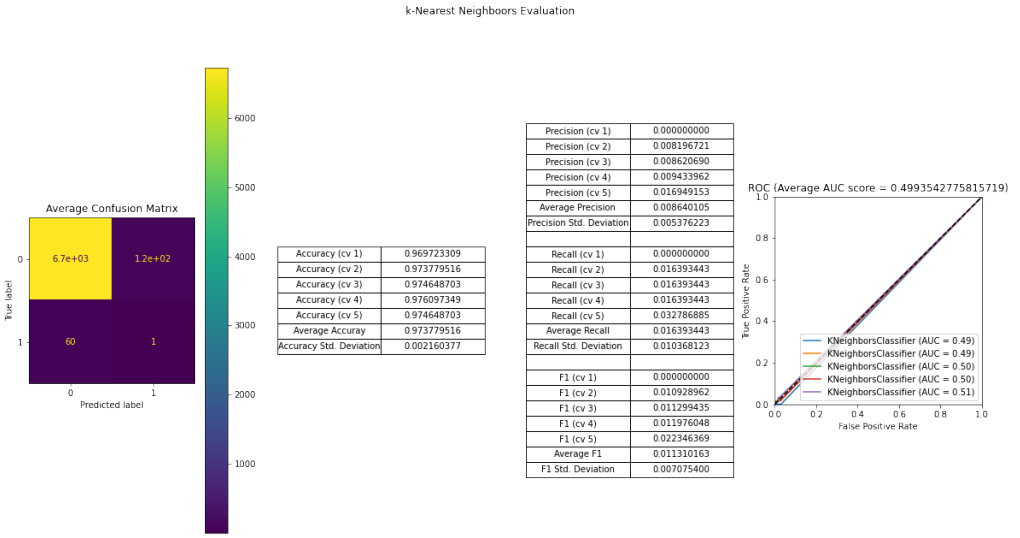
When trained on the balanced dataset with removed individuals, the k-Nearest Naighbors algorithms managed to perform worse than a random classifier (i.e. ROC AUC score lower than 0.5). But as soon as there are more individuals to learn on (with the dataset containing duplicated individuals), the algorithm performs very well (Figure 11). Using cross-validation, the average precision is close to 0.98 and the average recall is simply 1 (with very little deviation in each case). This means the classifier is able to make really good predictions (when it finds a defective individual it is correct about 98% of the times) while not missing any defective individual.
The latter might be very interesting for real world application as the company would rather manually check a item identified as defective and find out it is not defective rather than missing defective ones.



Accuracy (cv 1)	0.564826887
Accuracy (cv 2)	0.594524120
Accuracy (cv 3)	0.604519774
Accuracy (cv 4)	0.635375924
Accuracy (cv 5)	0.670143416
Average Accuray	0.613878024
Accuracy Std. Deviation	0.036044746

Precision (cv 1)	0.014516661
Precision (cv 2)	0.010721944
Precision (cv 3)	0.015959376
Precision (cv 4)	0.013470681
Precision (cv 5)	0.013181019
Average Precision	0.013569936
Precision Std. Deviation	0.001724986
Recall (cv 1)	0.721311475
Recall (cv 2)	0.491803279
Recall (cv 3)	0.721311475
Recall (cv 4)	0.557377049
Recall (cv 5)	0.491803279
Average Recall	0.596721311
Recall Std. Deviation	0.104507393
F1 (cv 1)	0.028460543
F1 (cv 2)	0.020986359
F1 (cv 3)	0.031227821
F1 (cv 4)	0.026305609
F1 (cv 5)	0.025673941
Average F1	0.026530855
F1 Std. Deviation	0.003385607

Figure 10: Naive Bayes Classifier Evaluation



Accuracy (cv 1)	0.969723309
Accuracy (cv 2)	0.973779516
Accuracy (cv 3)	0.974648703
Accuracy (cv 4)	0.976097349
Accuracy (cv 5)	0.974648703
Average Accuray	0.973779516
Accuracy Std. Deviation	0.002160377

Precision (cv 1)	0.000000000
Precision (cv 2)	0.008196721
Precision (cv 3)	0.008620690
Precision (cv 4)	0.009433962
Precision (cv 5)	0.016949153
Average Precision	0.008640105
Precision Std. Deviation	0.005376223
Recall (cv 1)	0.000000000
Recall (cv 2)	0.016393443
Recall (cv 3)	0.016393443
Recall (cv 4)	0.016393443
Recall (cv 5)	0.032786885
Average Recall	0.016393443
Recall Std. Deviation	0.010368123
F1 (cv 1)	0.000000000
F1 (cv 2)	0.010928962
F1 (cv 3)	0.011299435
F1 (cv 4)	0.011976048
F1 (cv 5)	0.022346369
Average F1	0.011310163
F1 Std. Deviation	0.007075400

Figure 11: k-Nearest Neighbors Evaluation

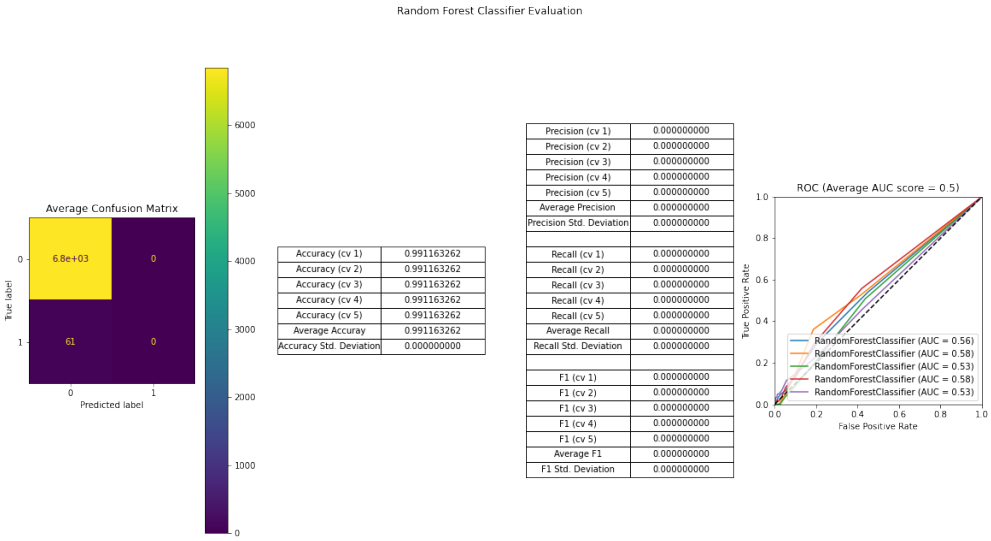


Figure 12: Random Forest Classifier Evaluation (with *class_weight* set to *"balanced"*)

4.4 Random Forest Classifier

Random Forest Classifier works by creating multiple Decision Trees on various subsets of the data, the decision of the forest for classification is simply the class that has been predicted by most of the trees [0]. A Decision Tree is built by finding the rule which devides the best the population of each node, this can be the rule that isolate most individuals from the same class (this method is called information gain and is based on the concept of entropy)[0]. A node cannot be devided when there are not enough individuals (in the node or in the to-be-created child-node), then the leaf is labeled with the class with the most representants on the node.

Random Forest Classifiers have a *class_weight* attribute in Scikit-Learn in order to address imbalance between classes[6]. However it seemed to perform worse than training the forest with the dataset containing duplicates (Figure 12). Figure 13 shows stunning results: sturdy precision and recall of 1 accross all cross validations.

While these results look great for now it remains to be seen if the forest is actually able to adapt to other errors. It is very likely that the classifier is now over-fitting both classes. By duplicating individuals, the model might learn that only these items are defective and will not be able to detect slightly different defective items.

4.5 Multilayer Perceptron

Multilayer Perceptron is part of the non-linear supervised learning branch. MLPs are characterised by iterative learning on the perceptrons present in the network layers [9]. Several arguments can be adjusted in order to modify the shape and parameters of the MLP:

- hidden_layer_sizes : number of neurons in each hidden layer
- activation : activation function for the hidden layer
- solver : solver for weight optimization

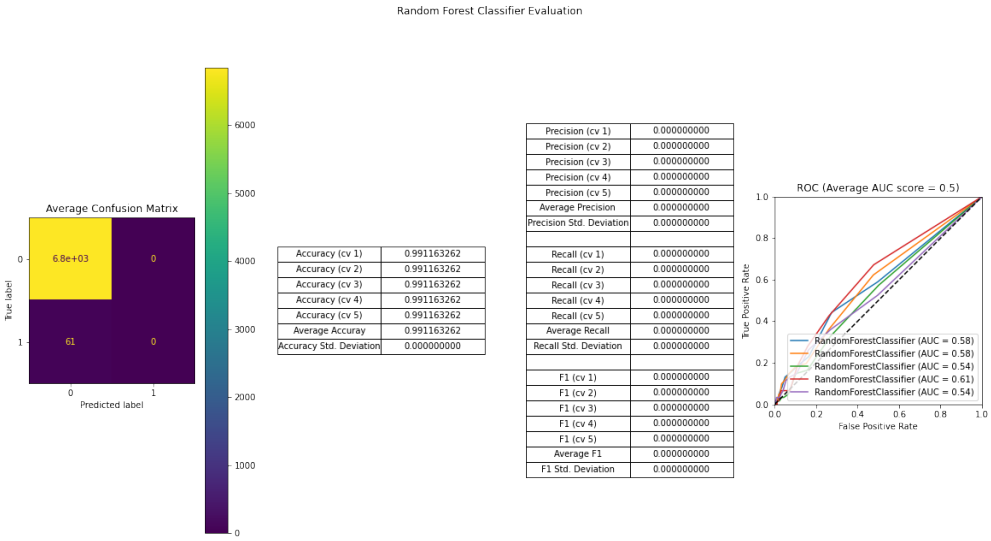


Figure 13: Random Forest Classifier Evaluation

- alpha : strength of the L2 regularization term
- batch_size : size of minibatches for stochastic optimizers
- learning_rate : learning rate schedule for weight updates
- max_iter : maximum number of iterations
- random_state : random number generation for weights and bias initialization
- early_stopping : early stopping indicator (terminate training when validation score is not improving)
- n_iter_no_change : maximum number of epochs to not meet tolerance for the optimization improvement

The MLP that gave the best results is composed of an input layer with 12 perceptrons and an output layer with one perceptron.

It is important to note that in the sklearn module, it is not possible to modify the cost function, which by default is the LogLoss function. The Logloss function minimizes the inverse of the log likelihood:

$$LL = -\frac{1}{n}\sum_{i=0}^{n}\left(y_i \log(p_i) + (1-y_i)\log(1-p_i)\right)$$

The loss is calculated for each batch, n is the value of the batch size, y_i is the output of the neural network and p_i is the associated probability.

There is no method to anticipate the most relevant hyperparameters. The only way is to test several combinations by changing only one hyperparameter at a time and to observe the performance of the resulting network on our data. The best parameters obtained during testing are presented in Figure 14.

Figure 15 shows the...

4.6 Novelty Detection

Anomaly or Novelty detection is a subset of machine learning particularly fitting to unbalanced class problems [5]. Knowing only the data and the estimated proportion of defects, the model will train itself and learn what

Hyperparameters	Selected value
hidden_layer_sizes	(200,150,100,50)
activation	hyperbolic tangent
solver	stochastic gradient descent
alpha	10^{-3}
batch_size	128
learning_rate	adaptive
max_iter	2000
random_state	1
early_stopping	True
n_iter_no_change	6

Figure 14: Best parameters for the MLP

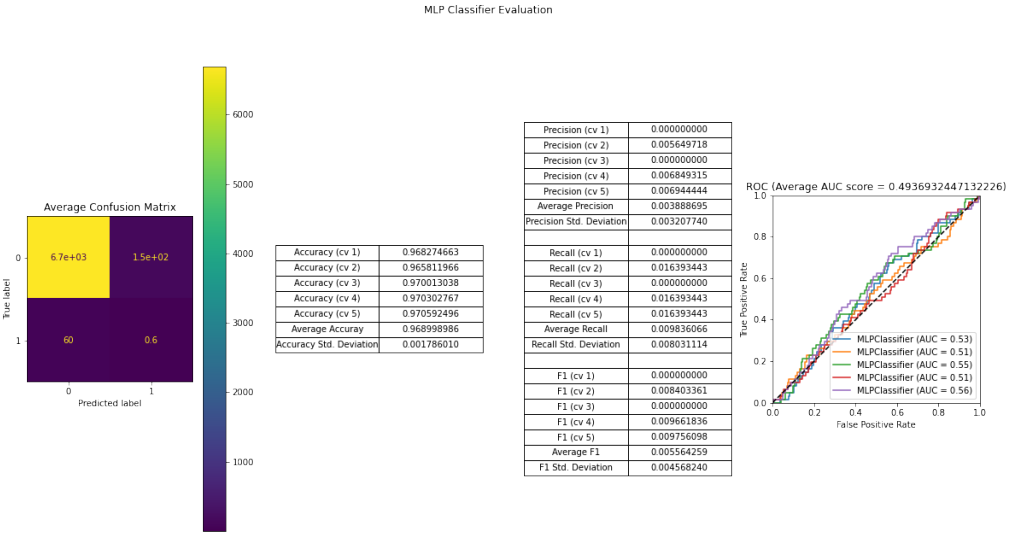


Figure 15: MLP Classifier Evaluation

a valid individual is supposed to look like [3]. It will then be able to classify new individuals: everything that is different than a valid item will be classified defective. Consequently, this means that the model is able to detect defects that are not represented in the training datset. This ability might be very be a advantge for this application as the model will stay relevant even when new defects occur. For instance, the tools on the assembly line will probably wear down leading to the apparition of new defects.

5 Model Fine Tuning

6 Conclusion

References

[1] Defect prediction on production line by valeo. https://challengedata.ens.fr/challenges/36. Accessed: 2023.

[2] Valeo. https://www.valeo.com/fr/. Accessed: 2023.

[3] Grab N Go Info. One class svm for anomaly detection — unsupervised machine learning. https://www.youtube.com/watch?v=0IkFnHpUUjE&ab_channel=GrabNGoInfo. Accessed: 2023.

[4] Scikit-Learn. Naive bayes. https://scikit-learn.org/stable/modules/naive_bayes.html. Accessed: 2023.

[5] Scikit-Learn. Novelty and outlier detection. https://scikit-learn.org/stable/modules/outlier_detection.html. Accessed: 2023.

[6] Scikit-Learn. sklearn.ensemble.randomforestclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Accessed: 2023.

[7] Scikit-Learn. sklearn.naive_bayes.gaussiannb. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html. Accessed: 2023.

[8] Scikit-Learn. sklearn.neighbors.kneighborsclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. Accessed: 2023.

[9] Scikit-Learn. sklearn.neural$_n etwork.mlpclassifier..$ Accessed: 2023.

Wikipedia. Decision tree learning. https://en.wikipedia.org/wiki/Decision_tree_learning. Accessed: 2023.

Wikipedia. k-nearest neighbors algorithm. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm. Accessed: 2023.

Wikipedia. Random forest. https://en.wikipedia.org/wiki/Random_forest. Accessed: 2023.