Machine Learning

Defect Prediction on Production Line

Team:
AIT BACHIR Romuald
ALLEMAND Fabien
FLORET Arthur
JARDOT Charles

École d'ingénieurs
Télécom Physique
Université de Strasbourg

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.

An in-depth look at the box-plots (Figure 2) reveals that there might be some outliers in the dataset.

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

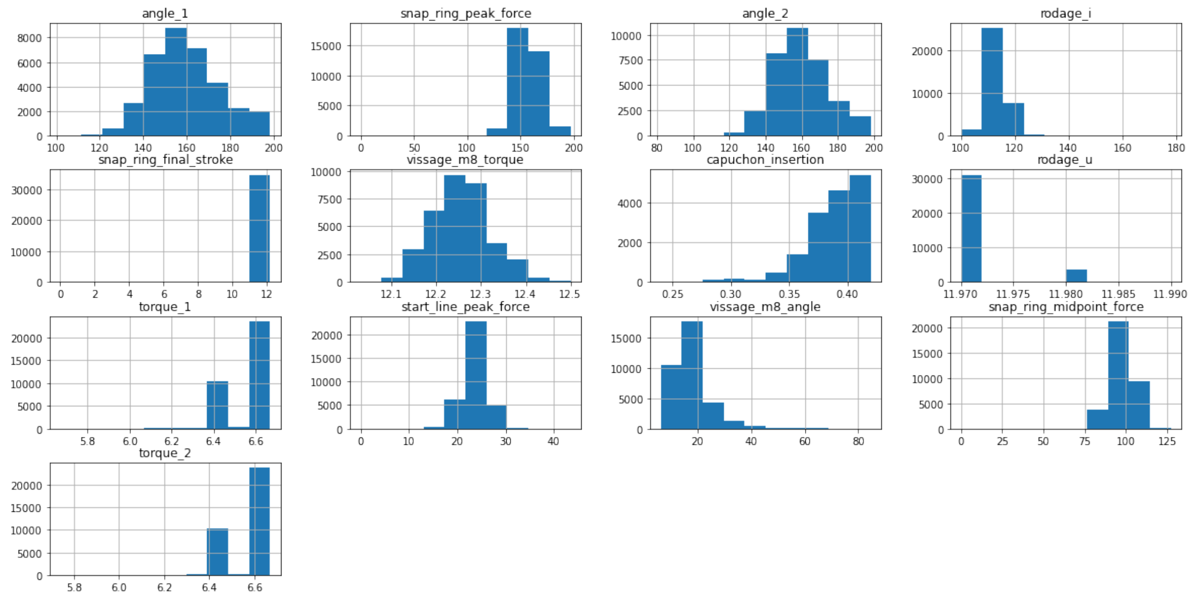


Figure 1: Distributions of the input features

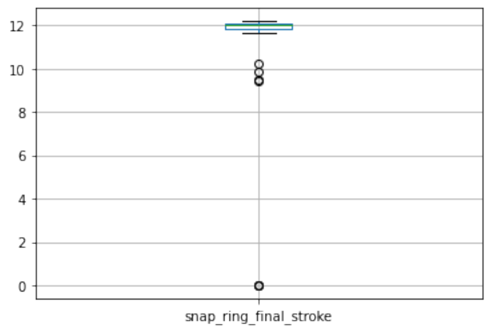


Figure 2: Box-plot of the *snap_ring_final_stroke* feature

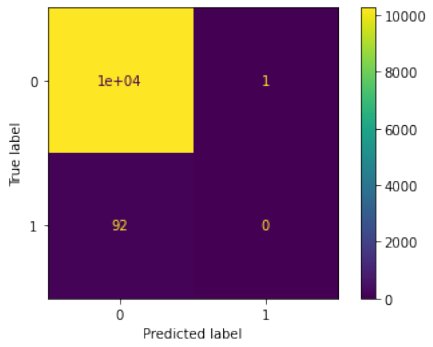


Figure 3: Confusion matrix of a k-Nearest Neighbors classifier with the base dataset

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 noticeables 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. Differents 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 shown in Figure 3, there are not enough defective individuals for the classfifier 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 removed).
Figure 4 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.
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

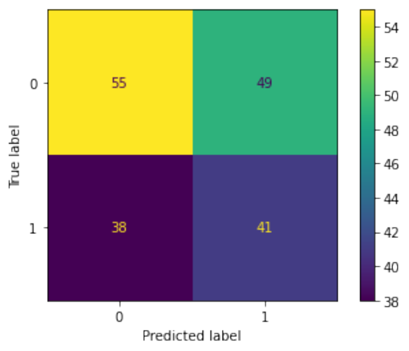


Figure 4: Confusion matrix of a k-Nearest Neighbors classifier with the dataset balanced by removing individuals

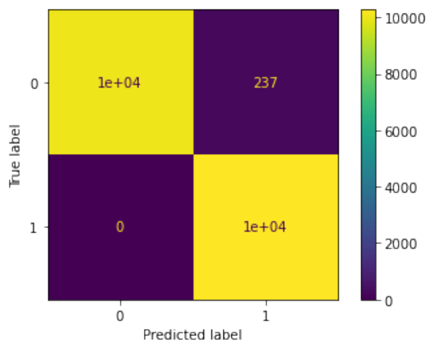


Figure 5: Confusion matrix of a k-Nearest Neighbors classifier with the dataset balanced by duplicating individuals

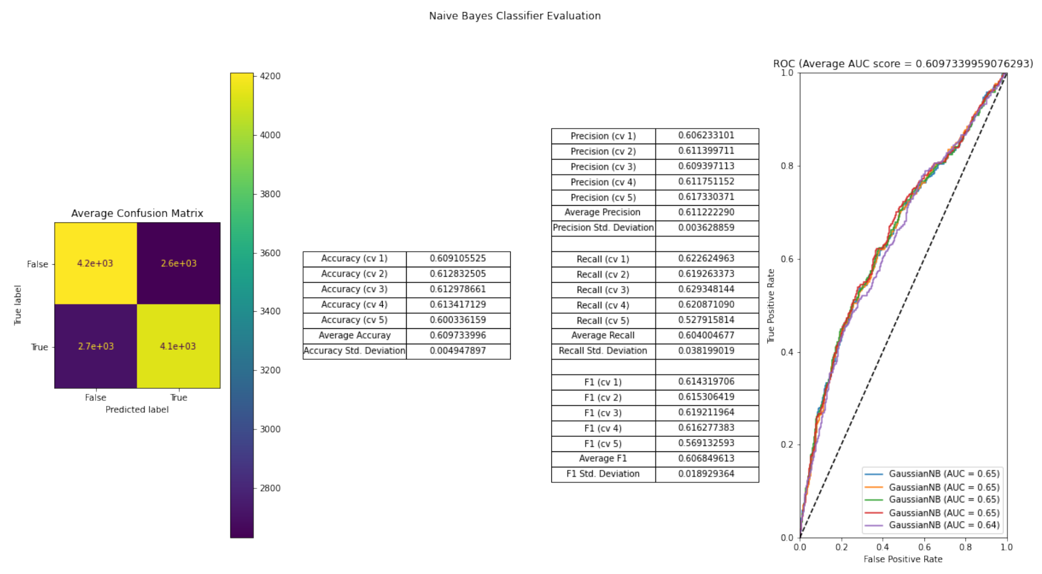
valid ones. This compensate the imbalance between the two classes without knowing anything about the features values. Figure 5 shows the effect on the classifier is significant.
Another way that has been implemented during this project is to slightly modify individuals during the duplication. This way 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 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 results are analysed using three different tools:

- Confusion matrix



Accuracy (cv 1)	0.609105525
Accuracy (cv 2)	0.612832505
Accuracy (cv 3)	0.612978661
Accuracy (cv 4)	0.613417129
Accuracy (cv 5)	0.600336159
Average Accuray	0.609733996
Accuracy Std. Deviation	0.004947897

Precision (cv 1)	0.606233101
Precision (cv 2)	0.611399711
Precision (cv 3)	0.609397113
Precision (cv 4)	0.611751152
Precision (cv 5)	0.617330371
Average Precision	0.611222290
Precision Std. Deviation	0.003628859
Recall (cv 1)	0.622624963
Recall (cv 2)	0.619263373
Recall (cv 3)	0.629348144
Recall (cv 4)	0.620871090
Recall (cv 5)	0.527915814
Average Recall	0.604004677
Recall Std. Deviation	0.038199019
F1 (cv 1)	0.614319706
F1 (cv 2)	0.615306419
F1 (cv 3)	0.619211964
F1 (cv 4)	0.616277383
F1 (cv 5)	0.569132593
Average F1	0.606849613
F1 Std. Deviation	0.018929364

Figure 6: Naive Bayes Classifier Evaluation

- Precision, recall and F1 score
- ROC AUC Score

These three tools are presented and explained in Section 5.
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 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 [3] (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:

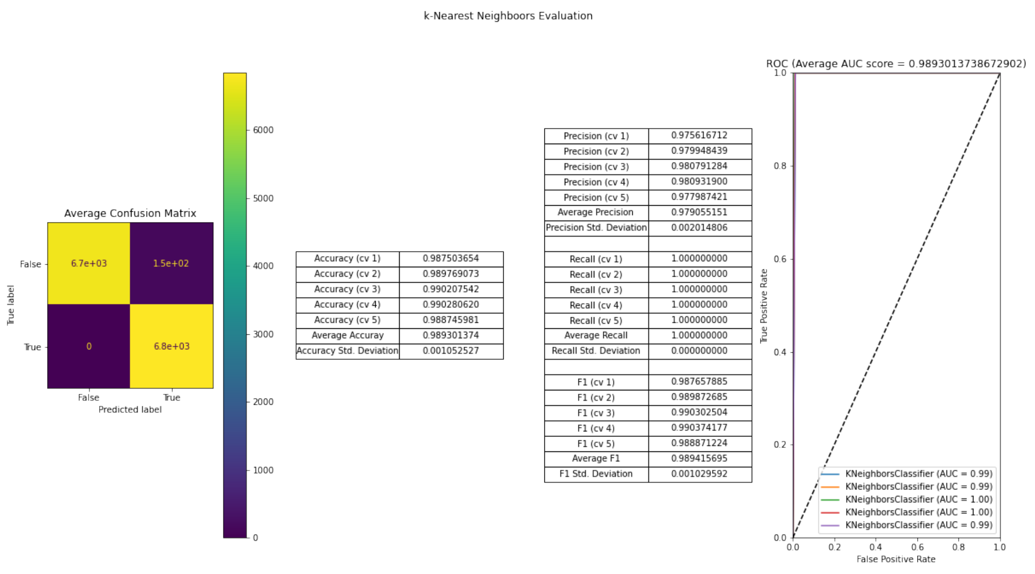
$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)} = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, ..., x_n)}$$

The following results correspond to a Gaussian Naive Bayes[4] 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 **??** shows poor results. This is expected because Figure 1 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*.

4.2 k-Nearest Neighbors

The k-nearest neighbors algorithm works by classifiying an individual by looking at its neighbors [5]. 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).



Accuracy (cv 1)	0.987503654
Accuracy (cv 2)	0.989769073
Accuracy (cv 3)	0.990207542
Accuracy (cv 4)	0.990280620
Accuracy (cv 5)	0.988745981
Average Accuray	0.989301374
Accuracy Std. Deviation	0.001052527

Precision (cv 1)	0.975616712
Precision (cv 2)	0.979948439
Precision (cv 3)	0.980791284
Precision (cv 4)	0.980931900
Precision (cv 5)	0.977987421
Average Precision	0.979055151
Precision Std. Deviation	0.002014806
Recall (cv 1)	1.000000000
Recall (cv 2)	1.000000000
Recall (cv 3)	1.000000000
Recall (cv 4)	1.000000000
Recall (cv 5)	1.000000000
Average Recall	1.000000000
Recall Std. Deviation	0.000000000
F1 (cv 1)	0.987657885
F1 (cv 2)	0.989872685
F1 (cv 3)	0.990302504
F1 (cv 4)	0.990374177
F1 (cv 5)	0.988871224
Average F1	0.989415695
F1 Std. Deviation	0.001029592

Figure 7: k-Nearest Neighbors Evaluation

4.3 Random Forest Classifier

4.4 Multi-Layer Perceptron

4.5 Novelty Detection

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

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

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