

Machine Learning

Defect Prediction on Production Line

Team:

AIT BACHIR Romuald

ALLEMAND Fabien

FLORET Arthur

JARDOT Charles



Abstract

Contents

1	Introduction	1
2	Data Analysis	1
2.1	Data Description	1
2.2	In-Depth Analysis	1
3	Data Preparation	3
3.1	Outliers	3
3.2	Balance Classes	3
4	Model Selection	4
4.1	Naive Bayes Classifier	5
4.2	k-Nearest Neighbors	5
4.3	Random Forest Classifier	6
4.4	Multi-Layer Perceptron	8
4.5	Novelty Detection	8
5	Model Fine Tuning	8
6	Conclusion	8

List of Figures

1	Distributions of the input features	2
2	Box-plot of the <i>snap_ring_final_stroke</i> feature	2
3	Confusion matrix of a k-Nearest Neighbors classifier with the base dataset	3
4	Confusion matrix of a k-Nearest Neighbors classifier with the dataset balanced by removing individuals	4
5	Confusion matrix of a k-Nearest Neighbors classifier with the dataset balanced by duplicating individuals	4
6	Naive Bayes Classifier Evaluation	5
7	k-Nearest Neighbors Evaluation	6
8	Random Forest Classifier Evaluation (with <i>class_weight</i> set to "balanced")	7
9	Random Forest Classifier Evaluation	7

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: $\text{Output} = f(\text{inputs})$

2 Data Analysis

The goal of this section is to take a first look at 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 corresponds 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_L_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 at the training data.

The training data consists of 13 measures (inputs features) made during production line on just shy of 35000 products. These measures correspond 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

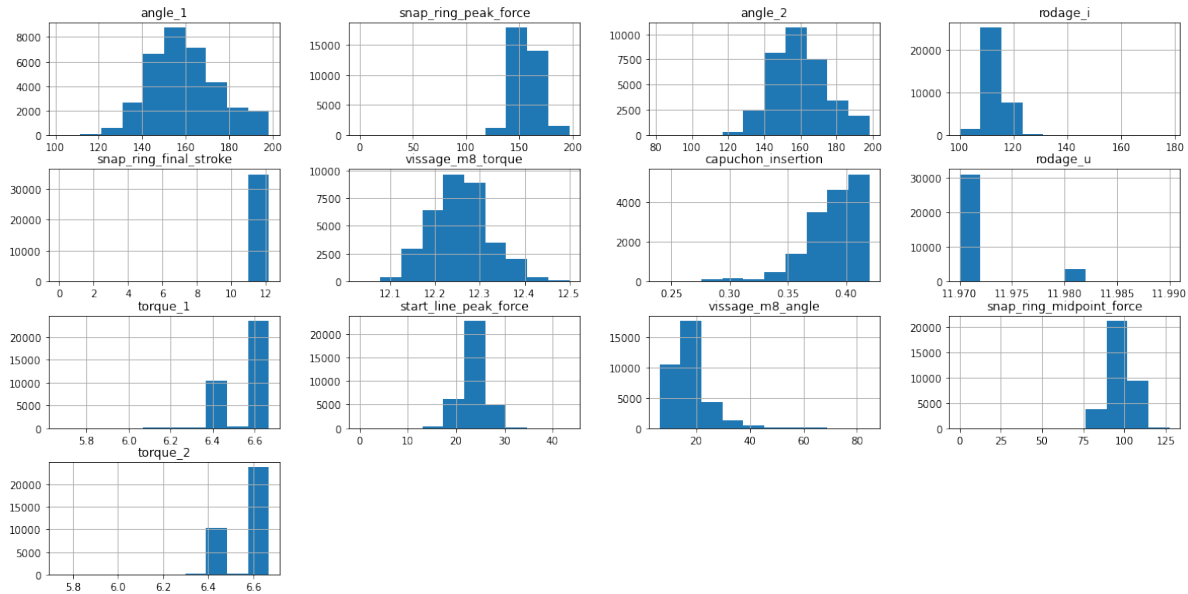


Figure 1: Distributions of the input features

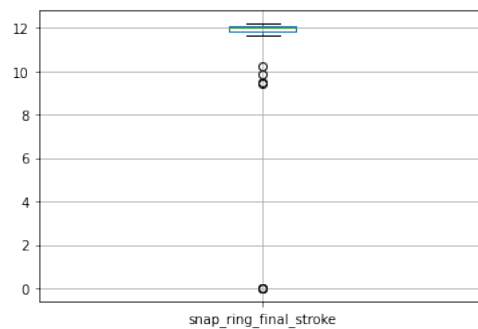


Figure 2: Box-plot of the *snap_ring_final_stroke* feature

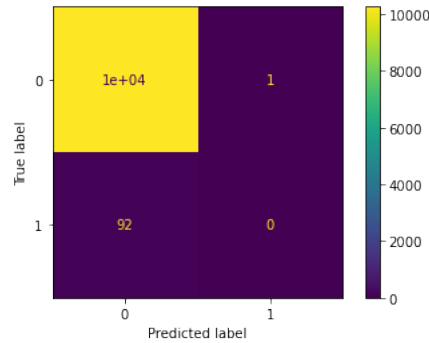


Figure 3: Confusion matrix of a k-Nearest Neighbors classifier with the base dataset

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 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 attempts 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 to 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 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 even though 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. Even though 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

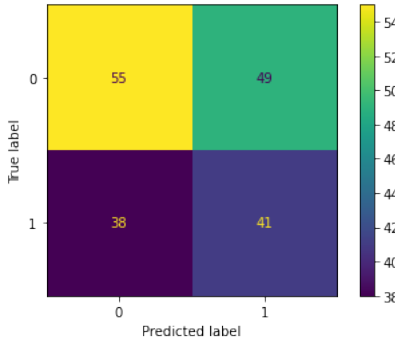


Figure 4: Confusion matrix of a k-Nearest Neighbors classifier with the dataset balanced by removing individuals

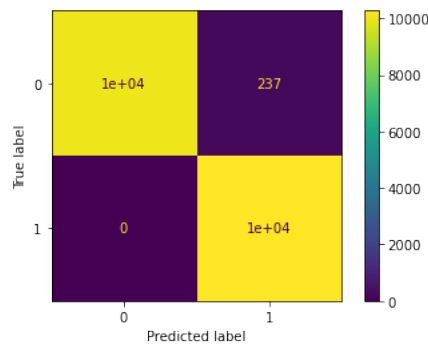


Figure 5: Confusion matrix of a k-Nearest Neighbors classifier with the dataset balanced by duplicating individuals

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 imbalanced 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

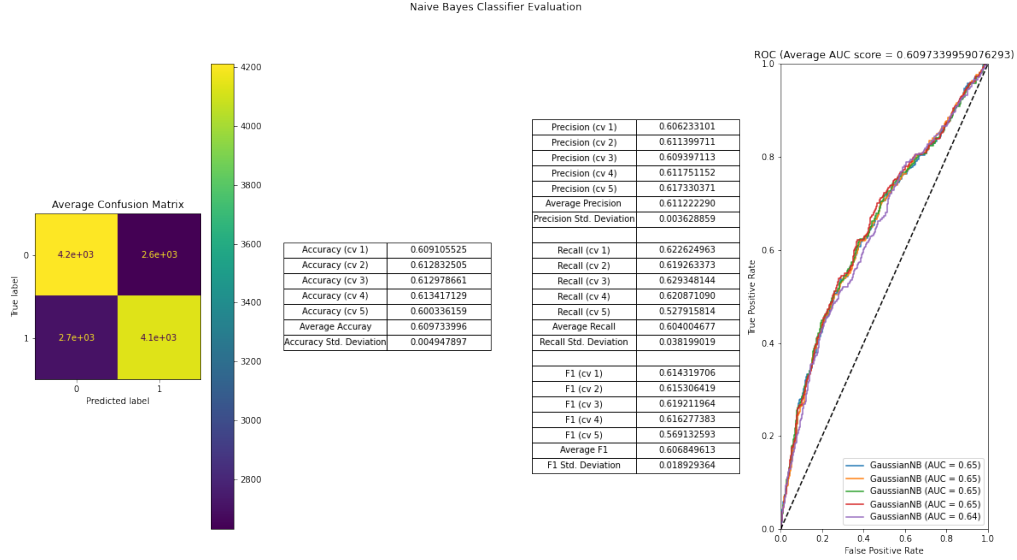


Figure 6: Naive Bayes Classifier Evaluation

- 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. $P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y)$) leading to the following equality:

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

The following results correspond to a Gaussian Naive Bayes[5] 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 classifying an individual by looking at its neighbors[8]. 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[6]).

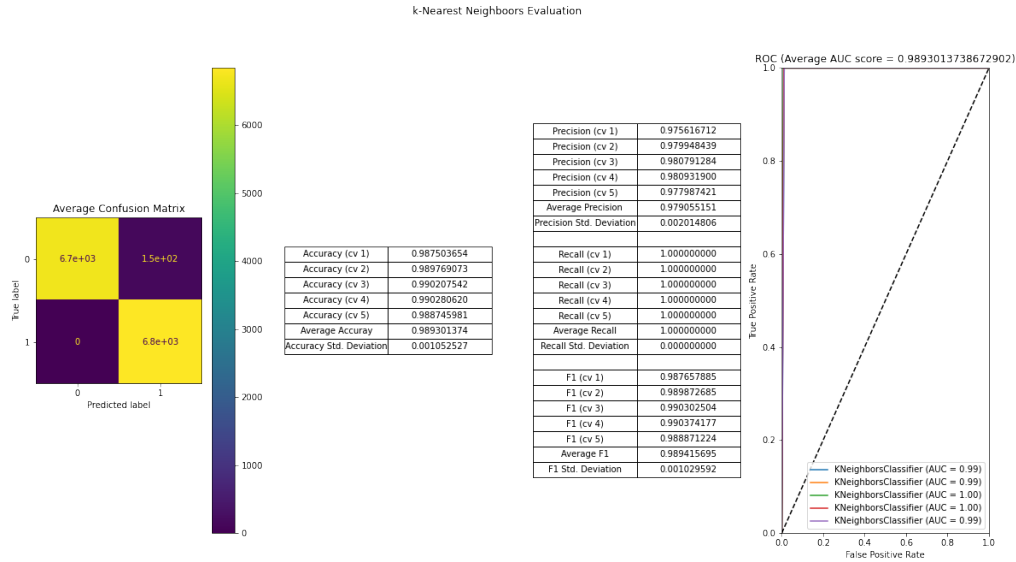


Figure 7: k-Nearest Neighbors Evaluation

When trained on the balanced dataset with removed individuals, the k-Nearest Naighbors algorithms managed to perform worse than a random classifier. But as soon as there is more individuals to learn on (with the dataset containing duplicated individuals), the algorithm performs very well (Figure 7). 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.

4.3 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[?]. A Decision Tree is built by finding the rule which divide 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)[7]. A node cannot be divided 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[4]. However it seemed to perform worse than training the forest with the dataset containing duplicates (Figure 8). Figure 9 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.

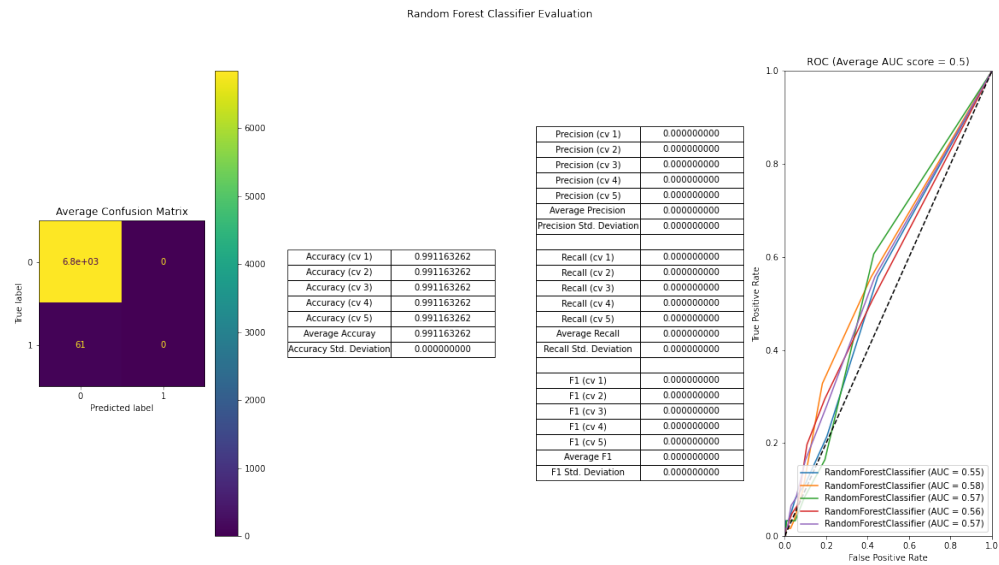


Figure 8: Random Forest Classifier Evaluation (with *class.weight* set to "balanced")

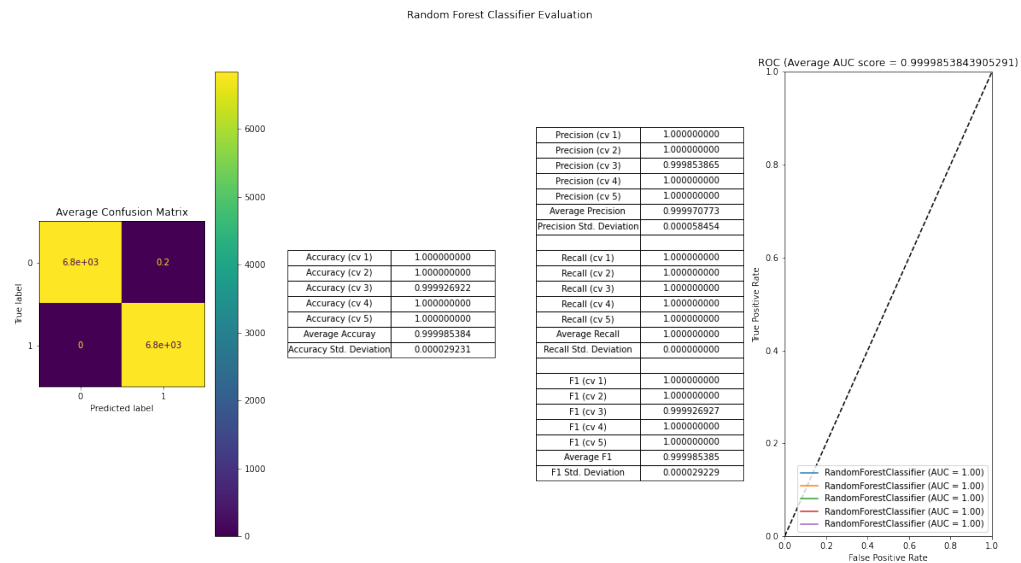


Figure 9: Random Forest Classifier Evaluation

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.ensemble.randomforestclassifier`. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>. Accessed: 2023.
- [5] Scikit-Learn. `sklearn.naive_bayes.gaussianmb`. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html. Accessed: 2023.
- [6] Scikit-Learn. `sklearn.neighbors.kneighborsclassifier`. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>. Accessed: 2023.
- [7] Wikipedia. Decision tree learning. https://en.wikipedia.org/wiki/Decision_tree_learning. Accessed: 2023.
- [8] Wikipedia. k-nearest neighbors algorithm. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm. Accessed: 2023.