The original version of this thesis contained confidential information of the industrial partner involved. To make publication possible, some passages had to be changed or deleted.

technische universität
dortmund

Bachelorarbeit

Machine Learning Methods for Optimized Industrial Product Acceptance Testing

Olga Akulinushkina
December 2020

Gutachter:
Prof. Dr. Katharina Morik
Dr. Helena Kotthaus

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (LS-8)
https://www-ai.cs.tu-dortmund.de/

Contents

Chapter 1

Introduction

1.1 Motivation

Manufacturing processes are highly automated and standardized. However, the newest information technology, such as machine learning, is often not utilized throughout the entire product lifecycle. The active integration of smart and innovative solutions in the manufacturing process, which is a key task in Industry 4.0, results in benefits ranging from increased visibility into operations, to substantial cost savings, to faster production times. Using sensors for data collection and cloud computing for storing and organizing makes data usable in the analysis and implementation of suitable machine learning models. This enables manufacturers to improve production, optimize operations, and gain a better understanding of the product and manufacturing phases. For example, the product test phase and decisions within it could be completely automated by machine learning algorithms to facilitate large production volumes.

Every hydraulic product analyzed in this thesis goes through a specified test to ensure high quality results. A machine learning model that can decide a priori whether a piece can or cannot pass a test can significantly shorten testing time. A reduced production time subsequently results in lower production costs.

1.2 Objectives

If a produced piece does not pass the conducted test, it receives the NOK label. The NOK products are then tested multiple times (see Fig. 1.1: red and green lines) to reduce the scrap rate. While testing products multiple times, various changes in the product can occur. For example, the product's temperature can increase, which would reduce the measured forces. As a result, a previously NOK product (see Fig. 1.1: red lines) can become an OK product (see fig. 1.1: green lines). This process reveals two classes of NOK product:

1. Product's label changes to OK.

2. Product's label remains unchanged.

Multiple tests are helpful for reducing the scrap rate of class 1 pieces. In contrast, tests for class 2 pieces should not be repeated. This thesis investigates whether an NOK piece can be classified as *changing* or *not changing* based on the data obtained during the multiple failed tests. Data available for the classification are present in form of a time series and extracted forces that are used for the test qualification. The extracted forces and time series should be investigated and preprocessed for the classification. The resulting model should be evaluated and validated with the other product types. The classifier should achieve the best possible results in the classification of both the classes.

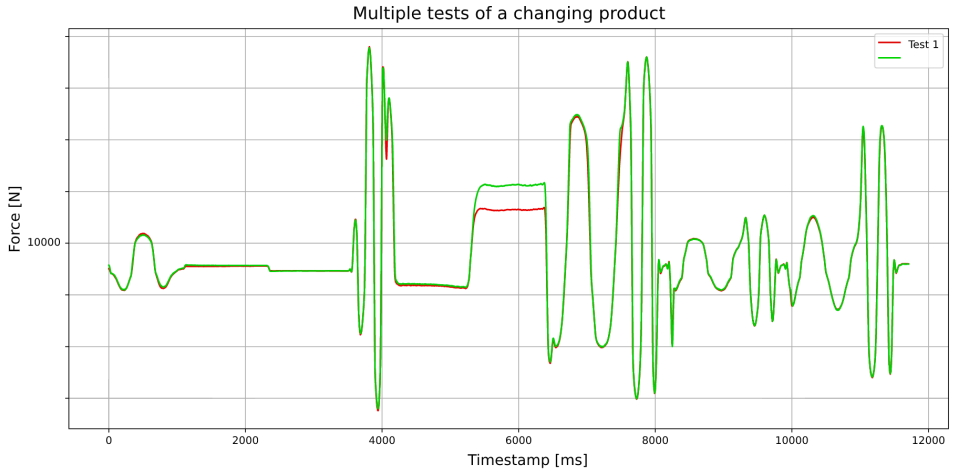

Figure 1.1: Force time series of the changing NOK product.

1.3 Structure

Chapter 2 introduces the specified test and measured forces and outlines the fundamental concepts used for feature extraction and selection, classification and noise correction.

Chapter 3 outlines the data preprocessing steps, that were followed to prepare the given samples for classification.

Chapter 4 presents the results of experiments with the given data. The present study explored different approaches for time series feature extraction, and evaluated and compared them.

The thesis ends with a conclusion in Chapter 5, that summarizes the results of the machine learning models implemented. Further, an overview of possible extensions and future work is provided.

Chapter 2

Background and Related Work

This chapter describes the specification of the test performed in the production, as values measured during this test served as the subject matter in the analysis. The final section provides an overview of related publications and demonstrates the fundamental concepts applied in the experiments conducted for this thesis.

2.1 Product Test

Every piece produced must comply with high quality standards. To ensure functionality, a specified product quality test is performed. At the end of the test, each produced piece is labeled *OK* or *NOK*. During this test, some mechanical operations are performed on a tested hydraulic product, resulting in a *univariate* force [N] *time series*. An example of the resulting signal, which are also called **raw data**, is shown in Fig. 1.1.

2.1.1 Test structure and extracted forces

For the product under investigation in this thesis, 16 interval forces are extracted from the raw data. These forces refer to the different mechanical conditions which the product is experiencing during the test. These extracted forces are used by a testing machine to define whether a tested piece is OK or NOK.

For all extracted forces, lower and upper limits are defined. A product is considered as NOK if at least one of these values is outside the given limits. Apart from these 16 forces, the raw data is not further used to define the test result. The nominal values for the limits are defined before the statistical distributions were known. Hence, the limits may need be adjusted at times, particularly during the early phases of the production of a new product.

2.2 Fundamental Concepts and related work

In this section, the fundamental concepts of experiments performed during this study are explained. The testing machine measurements are given in form of the *time series*.

2.2.1 Definition. Observation
A (time-dependent) observation is a vector $\overrightarrow{Z_t} \in Y^d$ with number of features $d \in \mathbb{N}$ and timestamp t. The value of the feature i is referred as $\overrightarrow{y_t}[i], 1 \leq i \leq d$.

2.2.2 Definition. Time series
A time series is a set of observations $Z_t \subset Y^d$, where Z_t is totally ordered by time $<_t \subseteq Y^d \times Y^d$, so $y_i <_t y_j \Leftrightarrow i < j$. A time series is called univariate if $d = 1$, otherwise multivariate.

2.2.1 Methods for feature extraction

In machine learning, feature extraction is a term for creating new features from the original data set.

The used time series is split into subsequences, named windows hereafter, using a windowing function.

2.2.3 Definition. Windowing
A time series Z_t of length T is split into windows $W_{i;l} = (x_i, \dots y_{i+l-1})$ of length $min(l, T-(1+k*s))$ using a windowing function with step value s and window index k:

$$windowing(Z_t, l, s) = \{W_{1;min(l,T-1)}, W_{(1+s);min(l,T-(1+s))}, \dots, W_{(1+k*s);min(l,T-(1+k*s))}\},$$

where $W_{1;l} = \{y_1, \dots, y_l\}$, $W_{(1+s);l} = \{y_{1+s}, \dots, y_{1+s+l-1}\}$, etc. If $s < l$, an overlapping with size $l - s$ occurs.

Variable	
l	Window size
s	Step size

Table 2.1: Adjustable hyperparameters of the windowing process.

Every described below approach for the feature selection was applied and investigated on the extracted windows (see details in Chapter 4).

2.2.4 Definition. Feature vector
Feature vector $\overrightarrow{f} \in X^{d_2}$ is a vector of features extracted from a time series $Z \subset X^{d_1}$. If the time series is splitted into the n subsequences $Z^* = \{Z_1, \dots, Z_n\}$, feature vectors $F = \{\overrightarrow{f}_1, \dots, \overrightarrow{f}_n\}$ are concatenated into one feature vector $\overrightarrow{f} = (\overrightarrow{f}_1, \dots \overrightarrow{f}_n)^T$

Extract aggregated values

The details that are seen in the time series of high resolution may contain invaluable information, representing the signal noise. Thus, the time series aggregation is a possible approach of handling high-dimensional data. Extraction of the statistical features was successfully used by Rodriguez and Kuncheva for the comparison of different classifiers in [23] and by Kampouraki, Manis and Nikou in [6] for the heartbeat time series classification. Stolpe, Blom and Morik used the time series aggregation for the classification of the real industrial data in [25]. Their study demonstrated that even a simple aggregation can be sufficient by achieving better results.

This approach is available in the Rapidminer [1] as operator "Extract Aggregates". Following values are extracted from the time series intervals:

2.2.5 Definition. Sum
Sum of the values of the time series Z_t is calculated as:

$$sum(Z_t) = \sum_{i=1}^{T} y_i.$$

2.2.6 Definition. Mean
Mean of the values of the time series Z_t is calculated as arithmetical mean:

$$mean(Z_t) = \frac{sum(Z_t)}{T}.$$

2.2.7 Definition. Maximum
Maximum of the values of the time series Z_t is defined as:

$$max(Z_t) = y_{max}, y_{max} \in Z_t \text{ and } y_{max} \geq y, \text{for all } y \in Z_t.$$

2.2.8 Definition. Minimum
Minimum of the values of the time series Z_t is defined as:

$$min(Z_t) = y_{min}, y_{min} \in Z_t \text{ and } y_{min} \leq y, \text{for all } y \in Z_t.$$

2.2.9 Definition. Median
Median of the values of the time series Z_t is defined as:

$$median(Z_t) = \frac{1}{2}(y^*_{\lfloor (T+1)/2 \rfloor} + y^*_{\lceil (T+1)/2 \rceil}),$$

where $(y^*_1, ..., Z^*_t) = Z^*_t$ and Z^*_t is acceding sorted time series Z_t.

2.2.10 Definition. First quartile
First quartile of the values of the time series is calculated as:

$$first\_quartile(Z_t) = median((y^*_1, ..., y^*_{\lfloor T/2 \rfloor})),$$

where $(y^*_1, \ldots, Z^*_t) = Z^*_t$ and Z^*_t is acceding sorted time series Z_t.

2.2.11 Definition. Third quartile
Third quartile of the values of the time series is calculated as:

$$third\_quartile(Z_t^*) = median((y^*_{\lceil T/2+1 \rceil}, ..., Z_t^*)),$$

where $(y_1^*, ..., Z_t^*) = Z_t^*$ and Z_t^* is acceding sorted time series Z_t.

2.2.12 Definition. Standard deviation
Standard deviation of the values of the time series is calculated as:

$$standard\_deviation(Z_t) = \sqrt{\frac{1}{T-1}\sum_{i=1}^{T}(y_i - mean(Z_t))^2}.$$

2.2.13 Definition. Kurtosis
Kurtosis of the values of the time series is defined as:

$$kurtosis(Z_t) = \frac{\mu_4}{standard\_deviation^4(Z_t)} = \frac{\mathbf{E}[X - \mathbf{E}[X]^4]}{standard\_deviation^4(Z_t)},$$

with μ_4 referring to the fourth central moment and $\mathbf{E}$ to the expected value calculated as:

$$\mathbf{E}[X] = \sum_{i=1}^{T} y_i p_i.$$

2.2.14 Definition. Skewness
Skewness of the values of the time series is defined as:

$$skewness(Z_t) = \frac{\mu_3}{standard\_deviation^3(Z_t)} = \frac{\mathbf{E}[X - \mathbf{E}[X]^3]}{standard\_deviation^3(Z_t)},$$

with μ_3 referring to the third central moment and $\mathbf{E}$ to the expected value.

Bag-of-SFA Symbols (BOSS) transformation

Time series can be transformed into symbolic sequences (words). Symbolic aggregation is a good approach in the noise and dimension reduction but may lead to information loss since values are being approximated. The original time series can be transformed into words by the Symbolic Aggregate approXimation (SAX) Algorithm proposed by Lin in 2003 [16]. SAX was used by Le Nguyen and others in [14] and applied by Matuschek in [17] and Gärtner in [12] in Rapidminer [1]. Another method for time series transformation is a Symbolic Fourier Approximation (SFA) proposed by Schärfer in [24]. In SFA, Discrete Fourier Transformation is first applied to the original values and then quantized into a word by using a defined alphabet.

2.2.15 Definition. Discrete Fourier Transformation
Discrete Fourier Transformation transforms the time series $Z_t = (y_1, ..., Z_t)$ of length T

from the time to the frequency domain of complex numbers $Y_f = (Y_{f1}, ..., Y_{fT})$, which is defined as:

$$Y_f = \sum_{n=1}^{T} y_i \cdot e^{-i\frac{2\pi}{T}nk} = \sum_{n=1}^{T} y_i \cdot [\cos\frac{2\pi}{T}nk - i \cdot \sin\frac{2\pi}{T}nk], \tag{2.1}$$

with k referring to frequency.

First $l << T$ Fourier coefficients are encoded into a word of length l. As higher frequencies of the DFT are associated with the signal noise, a transformed time series goes through the low pass filter in this step. [24]

The quantization of the Fourier coefficients is performed by using a Multiple Coefficient Binning (MCB). MCB bins the real and imaginary parts of the Fourier coefficients separately and maps them to the alphabet of symbols A of size S. The number of bins is S, accordingly. The result of the quantization step is a time series transformed into an SFA-word. [24] Fig. 2.1 demonstrates the encoding of the DFT in SFA words.

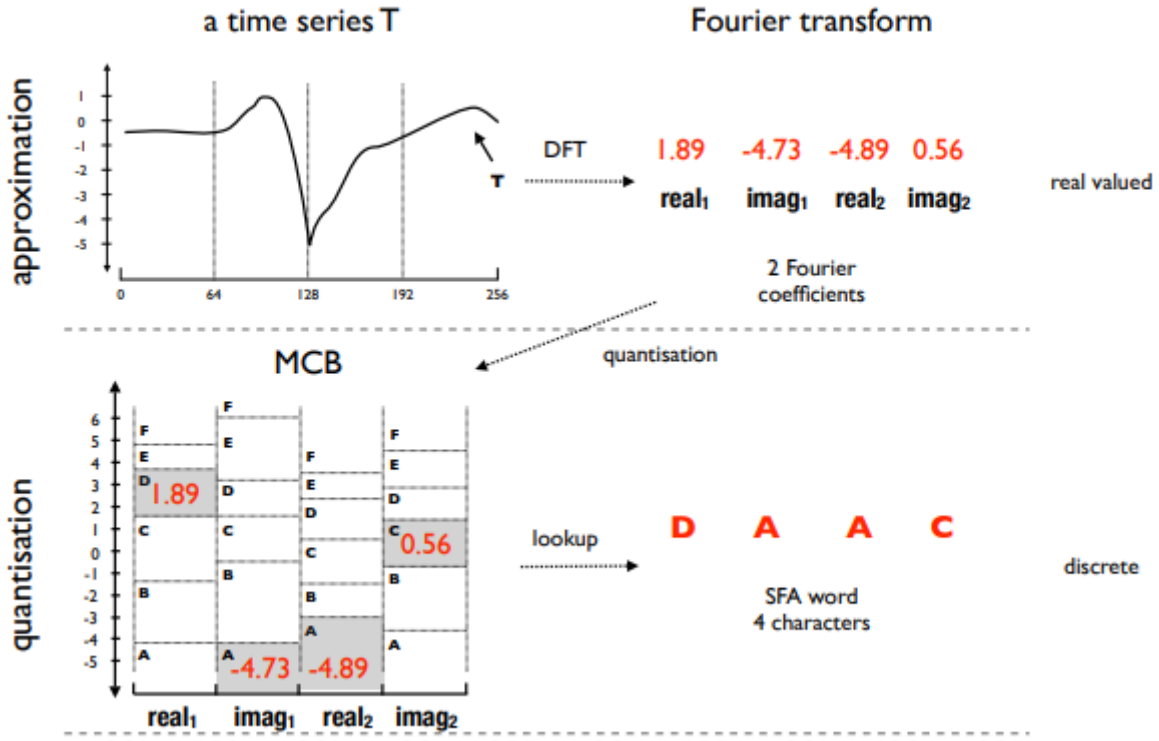

Figure 2.1: SFA: Discrete Fourier Transformation and MCB quantization. [24]

The BOSS model describes the given time series as a sequence of the SFA-words. SFA transformation is applied to the time series split into the overlapping windows. Extracted SFA-words will most likely be identical in the stable sections of the time series. The BOSS model performs *numerosity reduction* to avoid outweighed sections [24]. Multiple

consecutive appearances of the same SFA-word are removed from the sequence S , keeping only the first one:

$$S = \textbf{aab bbb} \text{ bbb bbb } \textbf{cca bcb} \text{ bcb bcb ...}$$

$$S' = \textbf{aab bbb cca bcb ...}$$

Lastly, BOSS constructs the histogram of the SFA-words frequency. Fig. 2.2 represents the visualization of the BOSS model steps. In contrast to the original publication [24], no *tf-idf* vectors are constructed for each class by an assumption that averaging the histograms will result in similar *tf-idf* vectors. Instead, the resulting frequencies are considered as features for every sample in future analysis.

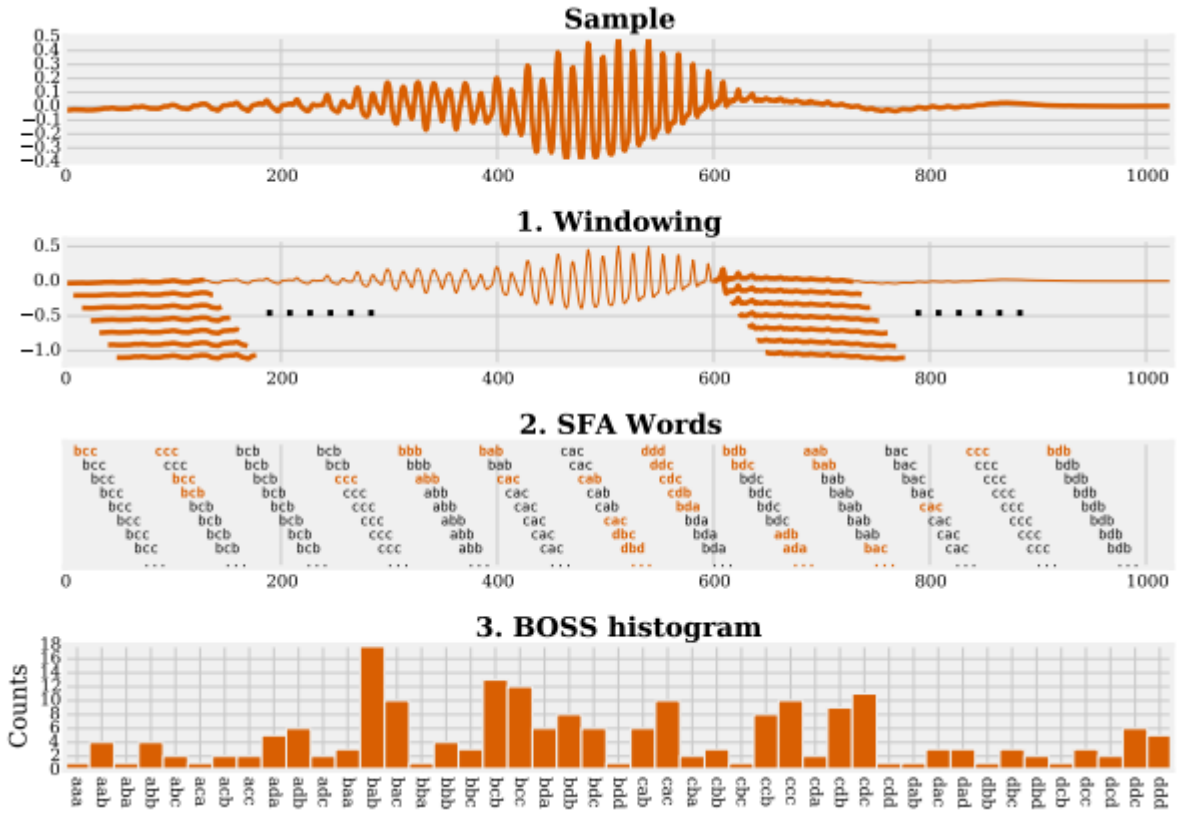

Figure 2.2: BOSS model visualized. [24]

Highest Peaks Extraction

Highest Peaks Extraction was one of the applied feature extraction methods in the publication of Mierswa and Morik [19]. For the analysis in this thesis, the Rapidminer [1] module "Peaks extraction" from the package "Time Series" was used. The algorithm of the highest peaks transformation implemented in the Rapidminer [1] is described in the following. [2]

The algorithm starts to work with the input time series. Subsequence used for the peak detection is referred to as *area* hereafter. First, the algorithm finds the global extremum

in the search area. Only the values above or below the average are candidates for the extremum. If there are no such values, the area is skipped.

Second, the method looks for the left and right end of the peak. These points of the time series must fulfill the following conditions: a) their values are not above or below the average; b) the relative change (decrease/increase for maximum/minimum) between the endpoint and its neighbor has to be larger than the *minimum change* per step.

After the peak and its left and right end are found, its value, position, type (minimum or maximum), amplitude, and width are saved as the new features. The algorithm continues to search for peaks in the areas left and right of the current area. The method is looking for N highest peaks (minimum and maximum) sorted by their amplitudes in the acceding order.

Variable	
N	Number of peaks (minimum and maximum together)
min_change	Minimum change between the values
$sloppy\_values$	Allowed number of values that do not fulfill the above condition

Table 2.2: Adjustable hyperparameters of the peaks extraction algorithm.

2.2.2 Methods for feature selection

Feature selection is a process performed to select the most relevant subset of the extracted features. A classifier can obtain better results even by using a smaller number of features. The runtime of the model training can also significantly decrease due to the feature reduction. For instance, highly correlated features can be removed in this step, since they can't improve the model's accuracy. Some classifiers, including random forests, are proofed to suffer from the correlation bias, which results in the confusing feature importance [26].

Genetic selection

Using genetic algorithms for the feature selection was proposed in 1992 by Leardi and others [21]. Morik and Mierswa used genetic feature exraction for the genetic feature extraction for the audio data classification in 2005 [19] and Matuschek in 2013 [17]. The genetic feature selection is available in Rapidminer [1] as a "Select Features (Evolutionary)" module. Genetic algorithms were inspired by the theory of evolution, inheriting its ideas, such as the advantage of the better combination of genes occurring as a mutation in the next generation against the worse genome. The whole population consisting of individuals with different genome tends to the best trough reproduction (*crossover*). Regarding feature selection, a feature refers to a gene, the combination of features refers to an individual or genome, and a group of such combinations represents a population.

The genetic algorithm consists of the following steps: features encoding, population initialization, crossover, mutation, evaluation, and selection. This study uses the Rapidminer [1] implementation [2],[18] of the evolutionary feature selection, which is detailed described in the following.

In the beginning, features are encoded as a binary code of zeros and ones. If a feature is selected for the subset with the probability p_i, it gets the code 1, otherwise, 0. The number of selected features F can be changed depending on the size S of the original feature set. The population of the given size P is initialized when P feature combinations (hereafter referred to as individuals) are generated.

In the first step, individuals are selected for the crossover with the probability p_c. There are numerous ways to perform a crossover between selected parents. In this study, a standard approach from the Rapidminer [1] was used: the resulting genome is combined from the first half of one parent and the second half of the other parent and vice versa. Given two parents as binary sets of feature codes f, $P_1 = \{f_{1,P1}, ..., f_{S,P1}\}$ and $P_2 = \{f_{1,P2}, ..., f_{S,P2}\}$, the resulting offsprings are $O_{1,P1,P2} = \{f_{1,P1}, ..., f_{\lfloor (S/2) \rfloor,P1}, f_{\lfloor (S/2) \rfloor+1,P2}, ..., f_{S,P2}\}$ and $O_{2,P1,P2} = \{f_{1,P2}, ..., f_{\lfloor (S/2) \rfloor,P2}, f_{\lfloor (S/2) \rfloor+1,P1}, ..., f_{S,P1}\}$. Both original and new combined individuals are kept in the population, doubling its size.

In the next step, mutation of genes is performed. Every individual goes through the mutation process with the probability p_m of each gene being flipped to the opposite value, 1 to 0 and vice versa. Original and mutated individuals are kept in the population for the evaluation, once again doubling its size.

The population size should now be reduced to its original size P by keeping only the best individuals. They are evaluated by the *fitness function*, which in this thesis is a cross-validated Naive Bayes classifier. Particularly, the resulting accuracy was used as the evaluation parameter for every individual.

After the population has been evaluated, the best group of individuals can be selected. One of the options can simply be a selection of the fittest individuals. Rapidminer [1] operator uses **tournament selection** [10] approach in this step. Tournament groups of the size T are randomly sampled (or bootstrapped) from the population and compared with each other. The winning group goes to the next generation. The tournament continues until the next generation of the size P is formed.

The described above steps, starting with the crossover, are repeated until the maximum number of generations G is reached or the evaluated accuracy does not improve for the given number of generations I.

Parameters $p_i, p_c, p_m, F, P, T, G, I$ can be adjusted.

Variable	
F	Number of selected features
P	Population size
T	Tournament group size
G	Maximum number of generations
I	Maximum number of generations without improvement
p_i	Probability of feature being selected for the subset
p_c	Probability of individual being selected for the crossover
p_m	Probability of mutation

Table 2.3: Adjustable hyperparameters of the genetic feature selection algorithm.

2.2.3 Classification methods

Naive Bayes classifier

Naive Bayes classifier [13] was used as an evaluation method inside of the genetic algorithm. This classifier was selected due to the feature set dimensionality. Using Random Forest or decision tree would increase the runtime of the feature selection step. Naive Bayes was applied by Abraham and others in [5] on the medical data set. Naive Bayes classifier combines from the Naive Bayes probability model with a decision rule. From the Bayes theorem, the probability that a sample X with features $\{x_1, ..., x_f\}$ belong in the class $c \in C$ is defined as:

$$p(c \mid X) = \frac{p(c) \cdot p(X \mid c)}{p(x)}. \tag{2.2}$$

The denominator is constant for every $c \in C$ because it does not depend on the class. The numerator equals to the joint distribution:

$$p(c) \cdot p(X \mid c) = p(c, x_1, ..., x_f) = p(x_1 \mid x_2, ..., x_f, c)...p(x_{f-1} \mid x_f, c)p(x_f, c)p(c). \tag{2.3}$$

[13]

By the "naive" assumption, all features in X and mutually independent and conditional on class c:

$$p(x_i \mid x_{i+1}, ..., x_f, c) = p(x_i, c). \tag{2.4}$$

[13]

Hence, the model is expressed as:

$$p(c \mid X) = \frac{1}{p(x)} p(c) \prod_{i=1}^{f} p(x_i, c). \tag{2.5}$$

The Naive Bayes classifier assigns class $\hat{c} \in C$ to the sample $\hat{X}$ by selecting the maximum posterior:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} p(c) \prod_{i=1}^{f} p(x_i, c). \tag{2.6}$$

Random forest

Random forest is an ensemble classification method of supervised learning, introduced by Breiman in 2001 [7]. Random forest is a combination of D random trees, which decisions are used for the classification. The original training set Q of size S is split into D new training sets $Q_1, \ldots, Q_D$ of the same size by the principle of **bagging**. $Q_1, \ldots, Q_D$ consist of the randomly selected elements from Q (*bootstraping*). Each bootstrapped training set does not contain every element from Q. Instead, some elements are randomly selected multiple times.

Samples that were not selected in the bootstrapping step are called **out-of-bag** (OOB) data, which is usually 36% [15] of the whole data (see proof in [8]). OOB data can be used for OOB error E_{OOB} calculation based on the training data. Error score shows how many test samples were assigned to the wrong class by the trained model.

Every random tree is constructed from a bootstrapped training set. In the original publication, Breiman proposed the random feature selection for every tree [7]. However, the python implementation used in this thesis allows every node of the tree to use all of the features and select the best one for split [3].

Random tree

A random tree is the building block of a random forest. In the random tree structure, leaves represent class labels, and branches represent conjunctions of features that lead to those class labels. Starting from the root, feature F is selected in every node to split data into subsets Q_s. Feature and its value used for the split are noted at the branch. A new node is created for every subset Q_s. The described process is recursively repeated. The algorithm stops if the tree reaches the given maximum depth, the size of Q_s is too small, or all of the leaves are pure. Leaves are pure if the node consists only of the samples of one class.

Features for the split can be selected by their information gain [8], gini impurity or accuracy. If the gini impurity is used as a criterion for the split, random tree searches for the feature with the greatest reduction of gini impurity. If gini impurity goes to 0, the chance of the randomly selected sample being misclassified, eventually, goes to 0 as well.

2.2.16 Definition. Gini Impurity
The Gini Impurity of a node n is the probability that a randomly chosen sample in a node would be incorrectly labeled if it was labeled by the distribution of samples in the node:

$$I_G = 1 - \sum_{i=1}^{J} p_i^2, \tag{2.7}$$

where L is the number of classes and p_i is the fraction of samples labeled as class $i \in \{1, \ldots, L\}$.

One tree can be vulnerable to the **overfitting**. Overfitting refers to the problem when the trained model performs well on the training data but fails to be accurate on the test set. To avoid bias in the decisions and assumptions, multiple random trees can be combined into an ensemble (random forest), bringing more variance to the model and making it more flexible.

Every tree in the ensemble votes for a class label. The final classification result of the whole ensemble is selected by the majority of votes (see Breiman, 2001 [7]). Python implementation of the random forest uses another approach. Classifiers are instead combined by averaging their probabilistic prediction [3]. The voting functions were compared by Brügge in 2011 in [8].

2.2.4 Noise correction methods

Real data sets often suffer from the wrong labeled samples in the training data set. Many different strategies can be applied to the classification with noisy labels. For example, examples that were most likely labeled incorrectly, can be removed from the data set (data cleaning). Boosting and bagging performed successfully in the classification with noisy labels. [11]. Another approach is to use the probabilistic models, e.g. Probabilistic Random Forest [22].

This thesis used the recent study and Python library cleanlab of Northcutt, Jiang, and Chuang [20]. The Confident Learning (CL) approach focuses on the identification of label errors in the data set. The publication also demonstrated consistent results in the classification of the noisy data sets.

Confident learning estimates the joint distribution between the noisy labels and the true latent labels [20]. CL uses the out-of-sample predicted probabilities $P_{k,i}$ and the vector of noisy labels $\tilde{y}_k$. $P_{k,i}$ are calculated by model θ, which produces a mapping $\theta : x \to p(\tilde{y} = i; x_k, \theta)$. Any model can be selected as θ, which affects the estimated probabilities and, hence, the CL.

The goal of the CL algorithms is to identify the noisy labels and improve the learning procedure. First, CL estimates the joint distribution $Q_{\tilde{y},y}$ of noisy labels $\tilde{y}$ and true labels y. To do that, the algorithm counts examples that are likely to belong to another class and captures it in the confident joint matrix $C_{\tilde{y},y} \in \mathbb{Z}_{\geq 0}^{m \times m}$.

2.2.17 Definition. Confident Joint [20]
Confident Joint $C_{\tilde{y},y}$ is formally defined as:

$$C_{\tilde{y},y}[i][j] := |\hat{X}_{\tilde{y}=i,y=j}| \text{ where}$$
$$\hat{X}_{\tilde{y}=i,y=j} := \left\{ x \in X_{\tilde{y}=i} : p(\tilde{y}=i; x_k, \theta) \geq t_j, j = \underset{l \in [m]: p(\tilde{y}=i; x_k, \theta) \geq t_1}{\operatorname{argmax}} p(\tilde{y}=l; x_k, \theta) \right\}$$

and the threshold t_j is the expected (average) self-confidence for each class

$$t_j = \frac{1}{|X_{\tilde{y}=j}|} \sum_{x \in X_{\tilde{y}=j}} p(\tilde{y}=i; x_k, \theta).$$

Informally, the confident joint estimates the set of examples $\hat{X}$ labeled as $\tilde{y}=i$ with large enough probability $p(\tilde{y}=i; x_k, \theta)$ to belong to class $y=j$, determined by a per-class threshold t_j.

With the given confident joint $C_{\tilde{y},y}$, the joint distribution $Q_{\tilde{y},y}$ can be estimated.

2.2.18 Definition. Joint distribution [20]
Joint Distribution $Q_{\tilde{y},y}$ is defined as:

$$Q_{\tilde{y}=i,y=j} = \frac{\frac{C_{\tilde{y}=i,y=j}}{\sum_{i \in m} C_{\tilde{y}=i,y=j}} \cdot |X_{\tilde{y}=i}|}{\sum_{i,j \in [m]} \left(\frac{C_{\tilde{y}=i,y=j}}{\sum_{i \in m} C_{\tilde{y}=i,y=j}} \cdot |X_{\tilde{y}=i}| \right)}$$

2.2.19 Definition. Latent prior [20]
Latent prior $Q_{y=j}$ is estimated as:

$$Q_{y=j} = \sum_i Q_{\tilde{y}=i,y=j}, \forall j \in [m].$$

2.2.20 Definition. Noise transition matrix [20]
The noise transition matrix $Q_{\tilde{y}=i|y=j}$ is estimated as:

$$Q_{\tilde{y}=i|y=j} = \frac{Q_{\tilde{y}=i,y=j}}{Q_{y=j}}, \forall i \in [m].$$

After all these steps, the data can be cleaned with any rank and pruning approach [20]. CL removes the error and reweightes the loss function for each class i $in[m]$ by:

$$\frac{1}{p(\tilde{y}=i|y=i)} = \frac{Q_y[i]}{Q_{\tilde{y},y}[i][i]}.$$

Chapter 3

Data preprocessing

The available NOK/OK labeled data of the tested was accumulated during several years. The NOK pieces were tested multiple times. The given data set contained the following information for *every test*:

1. Raw data force[N] time series with a resolution of 1 ms.
2. 16 extracted forces for the machine decision.
3. 32 upper and lower limits: two for each extracted force.
4. Metadata, such as DateTime, machine id, product id, and product.
5. Test result: OK or NOK.

Sixteen extracted forces and their limits were currently used by the testing machine to label every piece as OK or NOK.

The present study focused on one product in the machine learning model training. Other products were used for the validation to determine whether the implemented model could be successfully transferred to other production processes. The available data set of N samples was significantly reduced due the following cleaning and filtering steps.

3.1 Data filtering steps

Real industrial data sets often require a major amount of work put in the data preprocessing. Numerous issues, such as sensor failure, signal transferring error, data inconsistency, and lack of data, must be resolved before the data sets can be used for analysis. To classify a piece as *changing* or *non-changing*, the data set must consist exclusively of NOK pieces, also relabeled by the rule in table 3.1. Multiple tests of the same piece must be associated with each other and marked as first or last depending on the time the test was performed.

Label	First test	Last test
changing	NOK	OK
non-changing	NOK	NOK

Table 3.1: Labeling rule.

Every piece in the data set had an *id*, which could be stored in the table as NULL in case of a sensor failure. Tests with the missing id were removed because they could not be assigned to any piece. Production batches, in which all the products mistakenly had the same id were also removed.

An interesting aspect of the production process was the reuse of the id code if a NOK piece was removed from the production as scrap. In light of this, the id could not be used as a unique identification number because otherwise, a NOK piece could be incorrectly labeled as changing if its id was reused on an OK piece. To resolve this problem, the present study considered the *consecutive test runs* of the same piece as changing or non-changing, rather then the pieces themselves.

Some NOK pieces were tested again after some time and were found to change to OK under different circumstances (e.g., the first two tests had a result NOK→NOK, and two tests in an hour had different results NOK→OK). In these cases, both test run groups were removed from the data set to avoid noisiness in labels.

At times, mistakes occurred during the test process: some NOK pieces were not tested multiple times, while some OK pieces were tested repeatedly. These samples, along with any other OK pieces left, were removed from the data set. Following this, the data set contains only the changing and non-changing pieces.

In addition to the standard test procedure, some specific parts have to fulfill a more sophisticated and longer check. Both tests were stored in the same database. As only the standard tests are subject to this analysis, the longer ones are removed

Tests with missing values were handled in the same way (i.e., the whole test runs group were removed).

The number of tests performed for one piece varied from two to seven. This means, that some NOK pieces may not have been tested a sufficient number of times to change their state to OK. These samples can be confusing for the model, as their labels are incorrect. To keep the data consistent and to avoid this issue, the present study focused on the non-changing pieces tested exactly four times and the changing pieces tested up to four times, as this was the largest group available.

3.2 Preprocessing of the extracted forces

As discussed in 2.1.1, the given limits of extracted forces can vary over time. For this reason, the *absolute* values of the measured forces that should be considered as OK are different depending on the defined limits. To ensure that the model could work with any possible limits the absolute measured forces were transformed into the *relative* by calculating their **deviation from the limits** using the rule in the Table 3.2. If the measured force F_{meas} does not exceed the lower and upper limits (F_{lower}, F_{upper}), the deviation $d = 0$. If the measured force is bigger than the allowed maximum, $d > 0$. If the measured force is smaller than the required minimum, $d < 0$. Sixteen resulting deviation values, two for each interval, were considered features for the future analysis.

Measured value	Deviation
$F_{lower} <= F_{measured} <= F_{upper}$	0
$F_{measured} < F_{lower}$	$F_{measured} - F_{lower}$
$F_{measured} > F_{upper}$	$F_{measured} - F_{upper}$

Table 3.2: Deviation calculation rule.

3.2.1 Deviation groups

Every measurement interval was used to test different product components and their combination. Deviation in a certain interval can be a sign of failure of one of the used components. As the samples were not labeled for the root cause research, the tests were split into **deviation groups** by their deviation combination.

Every extracted force received a mark of "0" if $d = 0$, "+" if $d > 0$, or "-" if $d < 0$. As a result, every test received a string of 16 chars (e.g., "0 0 0 + 0 0 - + 0 0 0 0 0 0 0 0") as a deviation group name. This was later used to easily group similar tests, understand the classification results, and evaluate the implemented models.

3.3 Preprocessing of the raw data

Unlike extracted forces, raw data cannot be transformed into relative values due to the art the forces are being measured. Hence, it requires another filtering step, where the largest group with the same given limits is selected.

Different time series types were observed when the raw data were analyzed (see Fig. 3.1, 3.2).Time series of type A and B ran different test programs as shown in Fig. 3.1, 3.2. Pieces of type A first showed a positive force gradient while pieces of type B first experience a negative force gradient. As most of the products are tested by the program of type B

only this type was used for the analysis. Samples of type A were also removed from the data set of the extracted forces to keep the implemented models comparable.

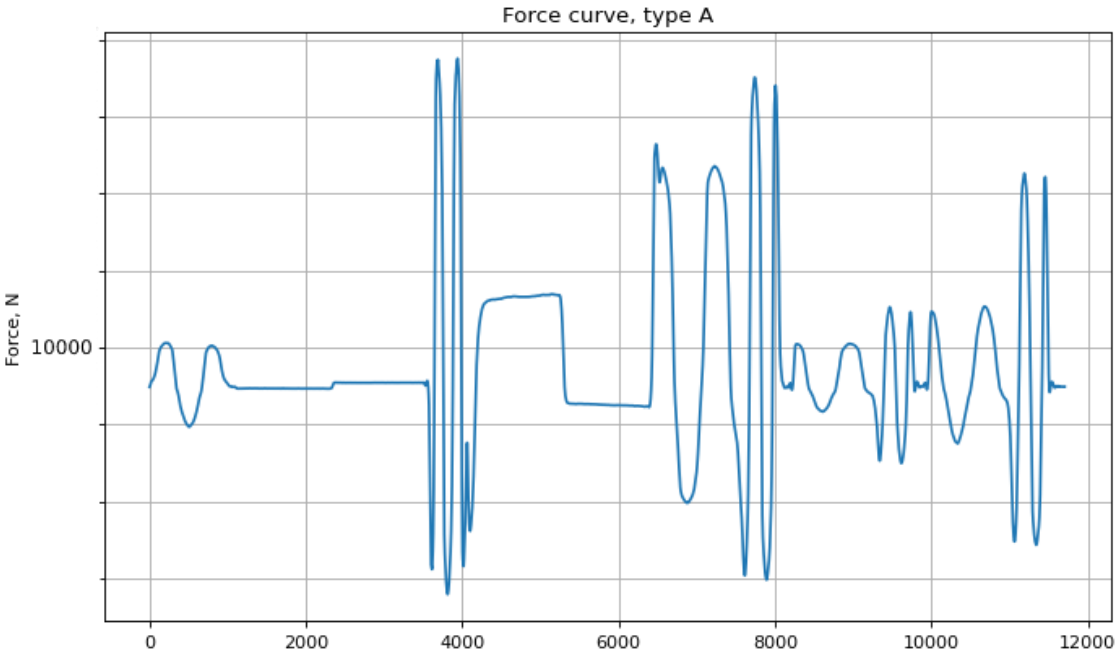

Figure 3.1: Time series of type A.

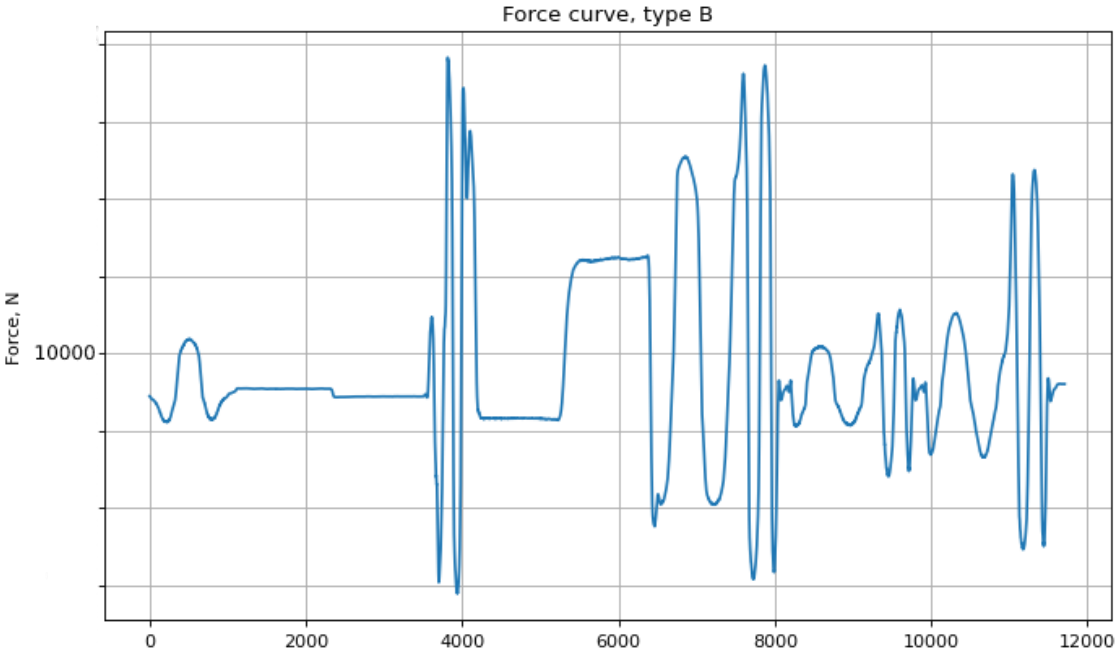

Figure 3.2: Time series of type B.

The raw data length varied from approximately 11,500 ms to approximately 12,300 ms. This difference came from either a late test start or the late test end when nothing

was occurring to the product, and constant values were being saved as a signal. Every time series was shortened to the length of the smallest time series because removing the constant end of the test does not affect the information gained.

Extraction intervals were marked in the data set. Hence, time series could be split into defined intervals, if necessary. The interval length could vary slightly. Intervals were shortened if they were used for the analysis.

3.4 Data set imbalance and noisiness

The preprocessing steps significantly reduced the number of available samples. Only 850 non-changing and 2300 changing pieces were left from the original data set of N samples. The non-changing class was poorly represented compared with the changing pieces. Its quantitative disadvantage made the data set imbalanced. Thus, we expect that the machine learning model might perform better in the classification of the changing pieces compared with non-changing ones.

The correct classification of the changing pieces is crucial for the production since the real OK pieces should not be removed from the test phase. The cost of a produced piece remains more valuable than possible time savings. On the other hand, the algorithm can only reduce the production time by identifying a non-changing piece and terminating its tests. High precision in the classification of the changing pieces is a primary requirement for the machine learning model. The goal of the experiments is to identify the model with the highest precision for the other class.

Since the data set is imbalanced, the overall model's accuracy is not considered as the most important evaluation value. The present study focused on F1-scores for both classes.

Both the whole non-changing class, and the failure root causes of the NOK piece may not have been represented sufficiently well to train and evaluate a robust model for the production. With no root cause label available, we could only refer to the deviation groups (see Section 3.2.1) to validate the stated assumption. There are 165 deviation groups in the data set of 2,856 samples; of them, 113 consisted of fewer than five samples. At times, some groups had only one example. These unique tests could be vulnerable in the classification if no similar sample was found in the training set.

Thirty-eight groups were mixed; these contain non-changing as well as changing products. Twenty of these mixed groups were small groups with fewer than five samples. The samples of the mixed groups were hard to distinguish even by considering at their raw data. Figure 3.3-3.8 illustrates four products from the same deviation group. All the products had a negative deviation in the specific interval (see 5500-6100 ms). The time series of the non-changing products (Fig. 3.3, 3.4) were similar during all four tests. There was only little difference in the problem interval (5500-6100 ms) or the rest of the time series. On the other hand, the changing products (see Fig. 3.5, 3.6) demonstrated improved mea-

surements by the second test. This could be seen in the problem interval 5500-6100 ms, where the second test had higher force values. The technical experts suggested that this improvement could be related to the specific problem that occurred in the first test and directly negatively affected the product's work. This specific problem can spontaneously resolve under continuous work. It is impossible to validate whether the root cause was correctly defined in this case. Again, the whole data set could not be labeled with the root causes in this bachelor thesis. Knowing the product's failure could help improve the classification of mixed groups.

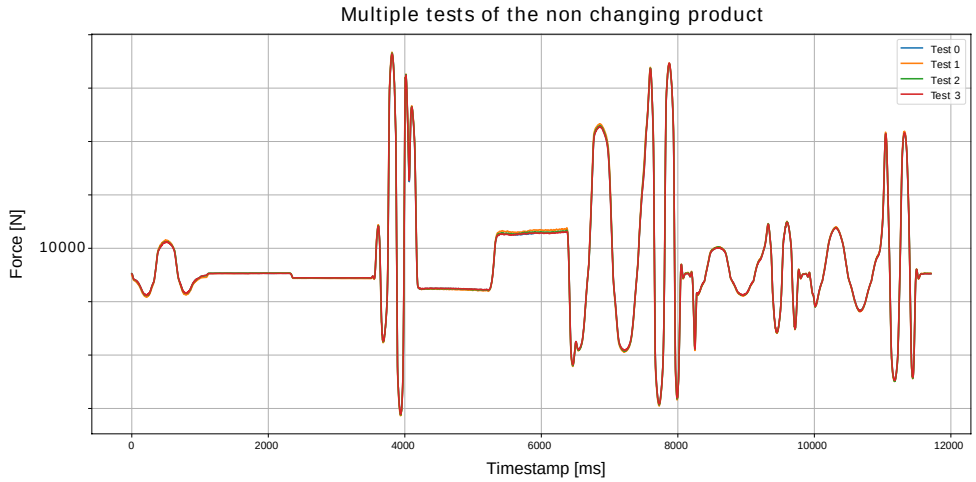

Figure 3.3: Not changing example 1.

Nevertheless, plotting all the described samples together (see Fig. 3.7 with the first test and Fig. 3.8 with the second tests) revealed some specific differences in the raw data. These differences were evident in the amplitude and peaks of the time series. Fourier transformation and peak extraction should be inspected as a possible classification and preprocessing strategy.

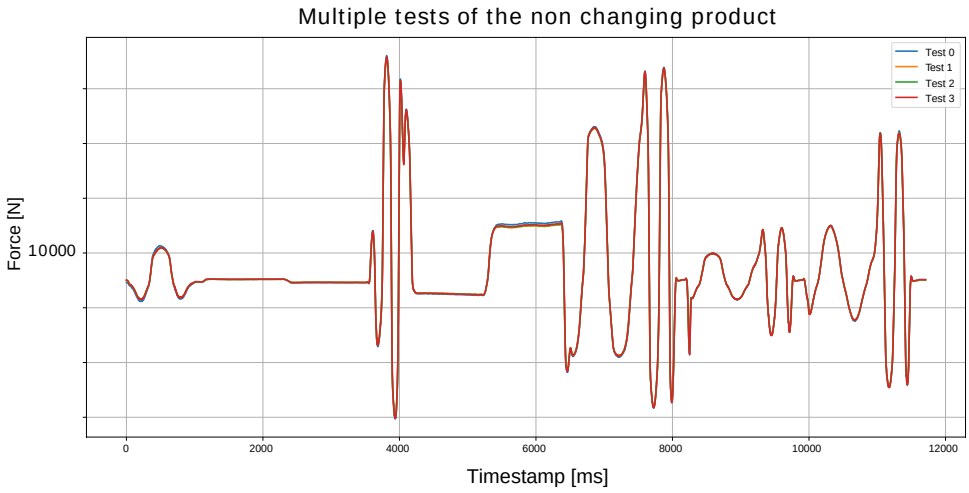

Figure 3.4: Not changing example 2.

Another approach would be to consider hard distinct samples as noise and perform noise correction. As previously stated, changing pieces require a different number of tests to improve to OK. We suggest that the data set might contained samples that simply needed more tests to change. Their mechanical characteristics and, hence, the raw data suppose to be the same as by the changing pieces. Therefore, correct classification with the wrong labels was impossible. Noisy labels should be either removed or flipped to the opposite ones.

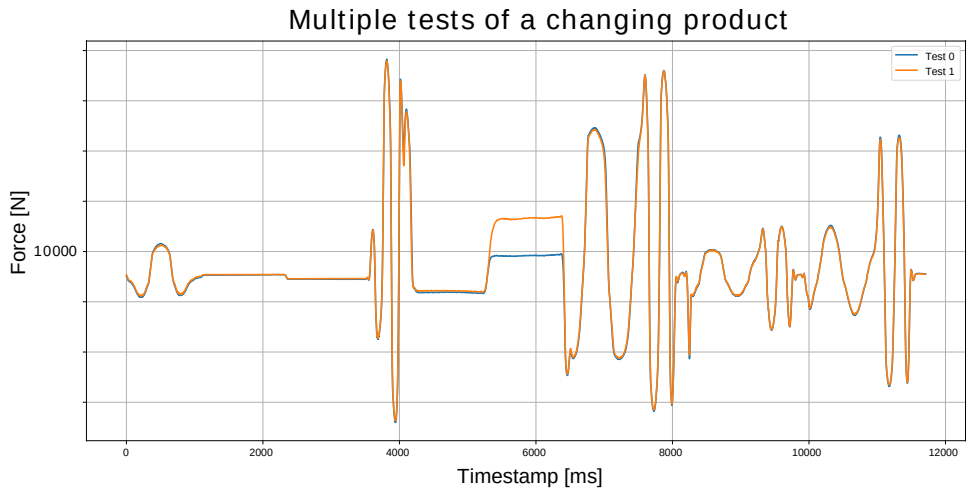

Figure 3.5: Changing example 1.

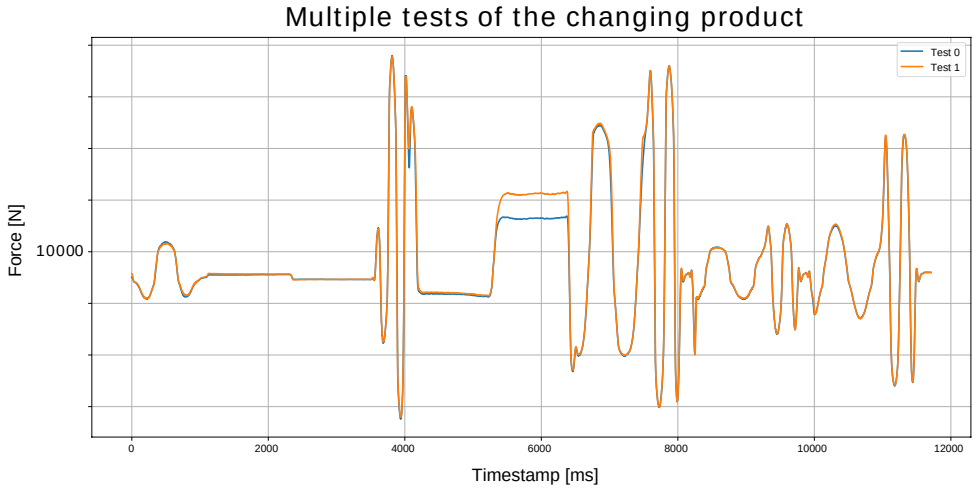

Figure 3.6: Changing example 2.

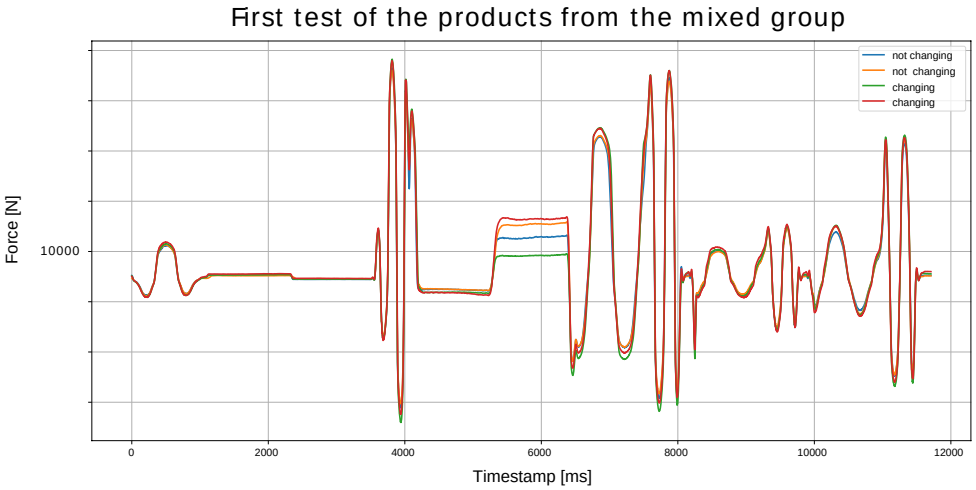

Figure 3.7: All examples (first test).

As the failure root cause of the failure was not known, we assumed that some samples could improve randomly under different circumstances. This randomness could be related to the meta-features, such as weather, date, production batch, and condition of the testing machine, which were not analyzed in the present thesis.

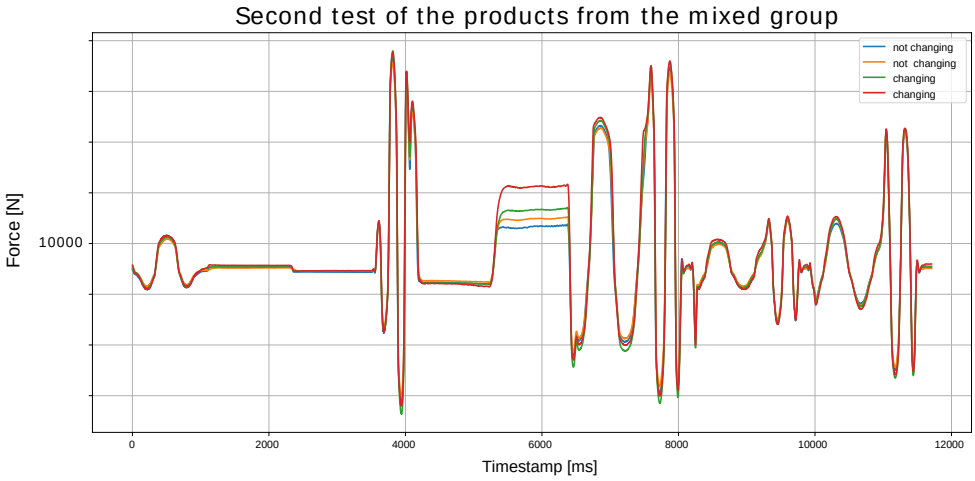

Figure 3.8: All examples (second test).

Chapter 4

Experiments

This chapter will presents the experiments on the classification and feature extraction methods and their results. Further, the motivation for every experiment is explained, and the results are validated.

4.1 Classification of the extracted forces

Changes in the given limits are the main reason for working with the extracted forces. As described in Chapter 3, the extracted forces were transformed to the relative values (see section 2.2.2). This transformation means that the classification is robust when applied to other products and, hence, to the different given limits. The requirements for every products can change in future. The model cannot be easily retrained on new data because it takes some time to accumulate them. A production-ready model should ideally work without regular supervision. The ability to use one model on all products would also be a major advantage. This is the main argument for investing more time into classification with extracted forces and seeking a working solution, even with a small number of features. Most problems and errors in the data were first revealed during the first experiments in the present study. Chapter 3 describes the cleaning steps performed. Information for the data understanding, which is the key to the evaluation and validation of the results, was also collected.

4.1.1 Experiment I: Classification of the first test results

We first assumed that the information gathered in the first test was sufficient to classify the piece as changing or non-changing. This would be the most time-saving approach for the testing process.

Experiment I setup:

Number of samples	825 non-changing, 2031 changing
Features	16 deviations in the first test
Environment	Python 3.7.7, pandas 1.0.5, sklearn 0.23.1
Classifier	Random Forest, number of trees 100, lists are pure criterion gini, confidence vote
Train/test split	80/20
Validation	shuffled 10-fold cross validation

Table 4.1: Experiment I: Classification of the first test results setup.

The hyperparameters of random forest were set default in all experiments. The data set was shuffled in all experiments due to the small number of available samples and the resulting instability in the model performance.

The results of the experiment are shown in Table 4.2.

	Not changing	**Changing**
Precision	0.75	0.87
Recall	0.65	0.92
F1-score	**0.70**	**0.92**
Accuracy	0.84±0.02	

Table 4.2: Experiment I: Classification of the first test results results.

As expected, in the first experiment, the model classified the changing pieces much better than the non-changing ones: 92% against 70% F1-score. The overall accuracy of the model remained by ≈84% with 10-fold cross-validation. Neither the main requirement nor the classification goal was fully achieved in the experiment.

The problems and concerns noted in section 3.4 were identified when investigating the results of the experiment. The random forest failed to classify the unique samples and mixed deviation groups (see Fig. 3.3-3.8). A lack of test samples and poor failure representation had a strong influence on the experimental results. The unique examples could not be correctly assigned to any class, as the training set did not contain similar samples.

Testing the product two times solved the mixed group problem. Although the first tests in these groups were too similar (see examples in Chapter 3, section 3.4, Fig. 3.3-3.8), the second test significantly differed and provided essential information on their class. Experiment II examines this hypothesis. Rather then testing the product two times, the noise correction approach was applied. In this case, some examples from the mixed groups were marked as noise in the data. This method is demonstrated in Experiment III.

The results can be interpreted in several ways. The data may require further filtering steps that are yet to be determined and may contain noisy samples or noisy labels that must be removed or reweighted. Nevertheless, the features selected in this experiment may be insufficient for correct and precise classification. As the possible benefit of this classification approach is valuable, attempts to improve this classification strategy would be worthwhile.

4.1.2 Experiment II: Classification of the first and second test results

Experiment II was performed as a possible improvement of Experiment I. We also expected to confirm whether the first test was sufficient for the classification with extracted forces. Hence, the experimental setup was the same as in Experiment I except for selected features. The first test deviations and their change to the second test were used.

Number of samples	825 non-changing, 2031 changing
Features	32 features: 16 deviations in the first test, their change in the second test
Environment	Python 3.7.7, pandas 1.0.5, sklearn 0.23.1
Classifier	Random Forest, number of trees 100, lists are pure criterion gini, confidence vote
Train/test split	80/20
Validation	shuffled 10-fold cross validation

Table 4.3: Experiment II: Classification of the first and second test results setup.

The results of the experiment are shown in Table 4.4.

	Not changing	**Changing**
Precision	0.85	0.91
Recall	0.77	0.94
F1-score	**0.81**	**0.93**
Accuracy	0.91±0.027	

Table 4.4: Experiment II: Classification of the first and second test results classification results.

The results of Experiment II were superior to Experiment I. The overall model's accuracy has improved from 84% to 91%, and the non-changing class had a higher F1-score (81% compared with 70% in Experiment I). These results partly confirmed the hypothesis that one test is insufficient for classifying specific product groups. However, most changing pieces (≈73%) changed their state to OK by the second test. Using the second test for the classification is, therefore, most suited to non-changing pieces. However, the F1-score

of the non-changing class remains far from perfect. In addition, the testing machine must spend more time testing the product twice. Despite the improved results, this experiment is likely unsuitable for production.

4.1.3 Experiment III: Classification of the first test results with noise correction

Experiment III was performed with the assumption that data contains noisy samples and needs more cleaning steps before machine learning methods can be applied (see Chapter 3, section 3.1, 3.4). The python library cleanlab (see Chapter 2, section 2.3.4) was applied to define the noisy samples. The library marked 11% of training examples as samples with possibly wrong labels. These were removed from the test set. The library removed noisy samples from the training set and reweighted the loss function (see Chapter 2, section 2.3.4). The estimated noise matrix was manually changed; therefore, the changing piece's label was not considered noise.

Number of samples	825 non-changing, 2031 changing
Features	16 deviations in the first test
Environment	Python 3.7.7, pandas 1.0.5, sklearn 0.23.1, cleanlab 0.1.1
%-Noise	11%
Classifier	Random Forest, number of trees 100, lists are pure criterion gini, confidence vote
Train/test split	80% of clean data + all noisy data /20% of clean data
Validation	shuffled 10-fold cross validation

Table 4.5: Experiment III: Classification of the first test results with noise correction setup.

	Not changing	**Changing**
Precision	0.95	0.95
Recall	0.85	0.98
F1-score	**0.90**	**0.97**
Accuracy	0.95±0.02	

Table 4.6: Experiment III: Classification of the first test results with noise correction classification results.

With the possible label error removed, the classification results improved. The overall accuracy and F1-scores of both classes subsequently exceeded 90%. However, it was unclear whether the noisy non-changing pieces had a chance to change, as they could be the outstanding pieces that the models from Experiments I and II were unable to classify.

By 10-fold cross-validation, the accuracy score remained stable by ≈95, but the F1-score of the non-changing class varied between 83% and 95%. The model demonstrated a varying ability to distinguish the problem class depending on which non-changing samples were included in the training set. Such sensitivity could be fixed by accumulating more data and training the model on a larger data set.

Due to a lack of data, it is difficult to predict how this model or any other model would perform under real production conditions. On the one hand, the model performed be better without noisy samples. On the other hand, these data filtering step might have hurt the real statistical distribution of the pieces tested. These results were difficult to validate without integration into the production process, as the representation of the failure types appeared weak, even in the original data set.

There were concerns about the data condition of the three experiments with the extracted forces. The classifiers did not perform perfectly, and the results of the selected approach improved by either adding more information or following additional filtering steps. Both these strategies appear to be the possible solutions for the classification task.

4.2 Classification of the raw data

It was assumed that raw data contain valuable information for the classification. Therefore, the classifier trained on extracted forces and the classifier trained on raw data could be combined in an ensemble. Alternatively, features extracted from the raw data could be combined with the extracted forces in one model.

4.2.1 Experiment IV: Classification of the aggregated values

The classification with extracted forces delivered promising results. Extracting more aggregated values from the whole time series and not only from the measurement intervals may improve the classification results further.

The whole time series was transformed into size 100 ms windows with step 50 ms with the windowing function. The Rapidminer operator "Extract Aggregates" was subsequently applied to every window. The extracted features went through the Rapidminer operators "Remove correlated attributes" and "Optimize Selection (evolutionary)". In total, 110 selected features were classified with the random forest.

Figure 4.1 shows the interval of 500 ms from 3.5s to 4s. The purple dashed lines highlight the local maximum point, and the black dashed lines mark the local minimum point. The distance between both maximum and minimum points equaled ≈200 ms. The window size was set as 100 ms to extract both of points as minimum or maximum values in the window with the operator "Extract aggregates".

Hyperparameters of genetic algorithm were set default except for the minimum number of attributes and the maximum number of generation. The minimum number of attributes

was selected regarding to the number of test intervals (16), and measured extracted forces (16). The maximum number of generations was set as 75 to allow the algorithm to select the best features but keep the runtime short.

Number of samples	825 non-changing, 2031 changing
Raw data	time series of length 11709 ms
Environment	RapidMiner 9.8
Windowing	Window size 100 ms, step 50 ms
Feature extraction	Extract Aggregates operator applied on every window
Feature selection	Remove correlated operator (correlation $>$ 0.95) Optimize Selection (Evolutionary) operator
Genetic algorithm	min. number of attributes 20 population size 15 maximum number of generations 75 p_i 0.9 p_m 1/n, n=number of features p_c 1 Naive Bayes classifier
Number of selected features	89
Classifier	Random Forest, number of trees 100, max. depth=200, criterion gini, confidence vote
Train/test split	80/20
Validation	shuffled 10-fold cross validation

Table 4.7: Experiment IV: Classification of the aggregated values setup.

	Not changing	**Changing**
Precision	0.80	0.87
Recall	0.68	0.93
F1-score	**0.73**	**0.90**
Accuracy	0.85±0.02	

Table 4.8: Experiment IV: Classification of the aggregated values classification results.

The results of Experiment IV, unfortunately, demonstrated that extracting more information from the raw data does not have any influence on the model's accuracy compared with Experiment I. The extracted values effectively represent the samples equally to the 16 extracted forces. This, again, can be interpreted as another sign of the label noisiness in the first test or poor data representation.

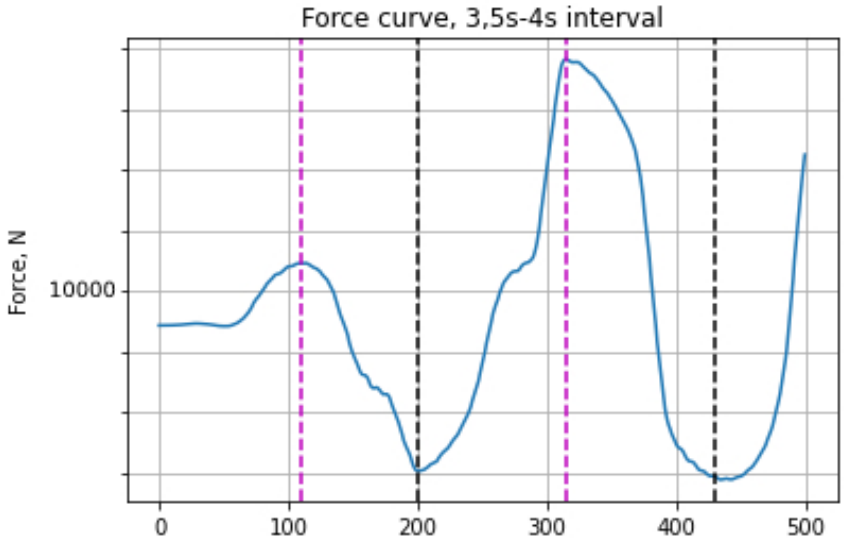

Figure 4.1: Interval from the example force time series.

Similar to Experiment II, we added additional information on the sample. Yet, the results of Experiment IV are worse. Using the second test as additional information brings more than extracting some values from the raw data of the first test. This shows that the first test results may be confusing and not differ enough to train a machine learning model.

4.2.2 Experiment V: Classification of the aggregated values with noise correction

As Experiment I and IV corresponded to each other, it was assumed that noise correction also helped to improve the results of Experiment IV.

	Not changing	**Changing**
Precision	0.96	0.94
Recall	0.79	0.99
F1-score	**0.87**	**0.96**
Accuracy	0.94±0.02	

Table 4.10: Experiment V: Classification of the aggregated values with noise correction classification results.

Yet again, we see the same pattern in the results. Removal of the noisy samples improved accuracy and F1-scores. Overall, the model achieves better results by classification of the changing pieces. Experiment V is another proof of the existence of the noisiness in the data. Now, when the whole time series is a subject to analyze, it remains impossible to get better results except for taking more filtering steps.

Number of samples	825 non-changing, 2031 changing
Raw data	time series of length 11709 ms
Environment	RapidMiner 9.8, Python 3.7.7, pandas 1.0.5, sklearn 0.23.1, cleanlab 0.1.1
Windowing	Window size 100 ms, step 50 ms
Feature extraction	Extract Aggregates operator applied on every window
Feature selection	Remove correlated operator (correlation > 0.95) Optimize Selection (Evolutionary) operator
Genetic algorithm	min. number of attributes 20 population size 15 maximum number of generations 75 p_i 0.9 p_m 1/n, n=number of features p_c 1 Naive Bayes classifier
Number of selected features	89
%-Noise	6%
Classifier	Random Forest, number of trees 100, max. depth=200, criterion gini, confidence vote
Train/test split	80% of clean data + all noisy data /20% of clean data
Validation	shuffled 10-fold cross validation

Table 4.9: Experiment V: Classification of the aggregated values with noise correction setup.

4.2.3 Experiment VI: BOSS-Transformation

Despite the previous unsuccessful results, we applied another method for the raw data transformation. The idea that aggregation of the time series is an incorrect approach for these data cannot be ruled out. As the principle of the piece and the test itself is based on mechanical oscillations, Fourier transformation represents a compelling method to experiment with.

BOSS Transformation was applied to every *test interval*, which were transformed into size 100 windows with the windowing function. The alphabet size was set to 5 for the detailed aggregation. BOSS used the first three Fourier coefficients to filter the signal noise out. The BOSS histograms were used as features with the classification with random forest.

Number of samples	825 non-changing, 2031 changing
Row data	time series split into defined test intervals
Features	BOSS histograms
Environment	Python 3.7.7, pandas 1.0.5, sklearn 0.23.1, pyts 0.11.0
Windowing	Window size 100 ms, step 1 ms
BOSS	word size 3, alphabet size 5
Number of features	1627
Classifier	Random Forest, number of trees 100, lists are pure criterion gini, confidence vote
Train/test split	80/20
Validation	shuffled 10-fold cross validation

Table 4.11: Experiment VI: BOSS-Transformation setup.

	Not changing	**Changing**
Precision	0.87	0.87
Recall	0.64	0.96
F1-score	**0.74**	**0.91**
Accuracy	0.87±0.02	

Table 4.12: Experiment VI: BOSS-Transformation classification results.

BOSS-Transformation slightly outperforms Experiment I and IV by only a few percent. However, the results keep showing the same tendency: $F_{1,\text{changing}} > F_{1,\text{non-changing}}$. Even using another approach for the raw data preprocessing did not demonstrate any significant improvement (accuracy 87% against 84% in Experiment I). Even the amplitude of the time series is unable to provide useful information for a certain classification, although it was expected.

4.2.4 Experiment VII: BOSS-Transformation with noise correction

The same strategy was followed, and the results of the BOSS transformation with and without noise correction were compared. The trend of delivering better classification results was expected to continue. The BOSS transformation was applied to the test intervals and not to the whole time series due to the faster runtime of the transformation.

Number of samples	825 non-changing, 2031 changing
Row data	time series split into defined test intervals
Features	BOSS histograms
Environment	Python 3.7.7, pandas 1.0.5, sklearn 0.23.1, pyts 0.11.0
Windowing	Window size 100 ms, step 1 ms
BOSS	word size 3, alphabet size 5
Number of features	1627
%-Noise	1%
Classifier	Random Forest, number of trees 100, max. depth=200, criterion gini, confidence vote
Train/test split	80% of clean data + all noisy data /20% of clean data
Validation	shuffled 10-fold cross validation

Table 4.13: Experiment VII: BOSS-Transformation with noise correction setup.

	Not changing	**Changing**
Precision	0.98	0.92
Recall	0.73	0.99
F1-score	**0.84**	**0.95**
Accuracy	0.93±0.02	

Table 4.14: Experiment VII: BOSS-Transformation with noise correction classification results.

As expected, compared with Experiment VI, the results improved by ≈5% in accuracy and the F1-scores of both classes significantly improved (+10% by non-changing and +4% by changing). This was achieved by pruning only 1% of the data (18 samples). The percentage of the detected noisy data decreased to 1% compared with 11% in Experiment I.

4.2.5 Experiment VIII: Classification of the extracted peaks

As we were unable to reach perfect results by aggregating or transforming time series, we looked for the details such as peaks. Hence, the operator "Extract Peaks" available in Rapidminer was used. The hyperparameters were set as follows: number of peaks 10, sloppy values 150, and minimum change 0.01. The sloppy values and minimum change were selected regarding the window size and expected peak width. Rather than extracting the small peaks referring to the signal noise, we focused on the wider peaks. As the resolution of the time series was 1 ms, the value change was relatively small. Hereafter, the minimum change was defined as 0.01 and verified empirically (see fig. 4.1). Window

size was set to 500 ms to reduce the run time. The smaller window size was not required in this experiment because the number of extracted peaks was set to 10.

Number of samples	825 non-changing, 2031 changing
Row data	time series of length 11709 ms
Features	peak position, type, amplitude, width, value
Environment	RapidMiner 9.8
Windowing	Window size 500 ms, step 500 ms
Extract Peaks	number of peaks 10 sloppy values 150 minimum change 0.01
Number of selected features	139
Classifier	Random Forest, number of trees 100, lists are pure criterion gini, confidence vote
Train/test split	80/20
Validation	shuffled 10-fold cross validation

Table 4.15: Experiment VIII: Classification of the extracted peaks setup.

	Not changing	**Changing**
Precision	0.87	0.80
Recall	0.37	0.98
F1-score	**0.52**	**0.88**
Accuracy	0.80±0.02	

Table 4.16: Experiment VIII: Classification of the extracted peaks classification results.

Experiment VIII demonstrated yielded poorer results than Experiments I-VII. The F1-scores of both classes did not achieve the expected level. This could have been due to incorrect hyperparameter selection or weak information gain from the peaks for the classification.

4.3 Summary of the experiments

Table 4.17 summarizes the experiments and their results. The best model is highlighted. Classification of the extracted forces with noise correction outperformed every other model. Next, the accuracy of the model was validated with the cleaned data from the other products. The difference between the products is absolute values of measured forces. The test structure remains the same. The validation set consisted of 628 not changing and 1,418 changing samples. The results are shown in Table 4.18.

	$F_{1,\text{not changing}}$	$F_{1,\text{changing}}$	Accuracy
Extracted forces (first test)	0.70	0.92	0.84
Extracted forces (first and second test)	0.81	0.93	0.91
Extracted forces (first test) with NC	**0.90**	**0.97**	**0.95**
Aggregated values (raw data)	0.73	0.90	0.85
Aggregated values (raw data) with NC	0.87	0.96	0.94
BOSS histograms	0.74	0.91	0.87
BOSS histograms with NC	0.84	0.95	0.93
Peaks	0.52	0.88	0.80

Table 4.17: Experiments results (mean values) summary.

	Not changing	**Changing**
Precision	0.94	0.98
Recall	0.95	0.97
F1-score	**0.95**	**0.97**
Accuracy	0.96	

Table 4.18: Validation results, extracted forces (first test).

As expected, the model with noise correction could also be applied to other products. However, if the validation set was not cleaned, the performance of the model remains at 86%, F1-scores at 80% and 89% (see tab. 4.19). Thus, how the trained model will perform on real data in the production remains a concern. The data noisiness should be investigated further, and the non-changing pieces that the classifier mistakenly labeled as changing should be analyzed. These pieces may require more tests to change, and the model may be correct in its decision.

	Not changing	**Changing**
Precision	0.82	0.87
Recall	0.77	0.90
F1-score	**0.80**	**0.89**
Accuracy	0.86	

Table 4.19: Validation results with noisy validation set, extracted forces (first test).

Chapter 5

Summary and Futurework

Chapter 5 summarizes achieved results, while the conclusion suggests possible improvements and next steps for future analysis.

5.1 Summary

Before any machine learning method could be applied to the provided data, multiple preprocessing steps had to be performed. Defining and executing these, the number of available samples was significantly reduced. Data preprocessing and the first experiments delivered a better understanding of the production process, product itself, and physical processes behind the tests performed. The results raised specific concerns about the quality of the available data set and uncovered possible directions for the research.

Missing root cause labels may make classification more difficult for the mixed groups of similar samples. The unique pieces may also be classified wrong as no similar samples are found in the training set. Some changing and non-changing pieces may have similar mechanical characteristics but differ in the number of tests required to change. If so, some samples in the available data may have an incorrect label and require label noise correction.

Two possible approaches for the introduced classification task were investigated in the present thesis: analysis of the extracted forces and the raw data.

Although the results of all classification strategies differed slightly from each other, the following trends could be observed:

- The performance in the classification of the changing pieces was better than in the other class due to the data set imbalance.
- Every classification and feature extraction technique failed to exceed 90% accuracy value without noise correction.
- Information gathered during the second test improved the performance of the classifier.

- Noise correction boosted every model up to the 95% accuracy score even with the different number of noisy samples found.

The model with the best performance (extracted forces, first test, noise correction) was validated with the data from the other products and demonstrated the same stable results.

5.2 Conclusion

The goal of the thesis was achieved by applying various machine learning methods to the provided data set and evaluating the results. The model performance was validated with the additional data set.

The achieved results showed that the data set requires further cleaning steps. Rather than focusing on different classifiers or feature extraction strategies, noise correction methods should be investigated and validated. For instance, this study did not attempt to use any clustering strategies to find outliers and validate the noisy samples. Outliers are samples which are similar to some group but have a different label. For example, optimally tuned improper maximum likelihood estimator (OTRIMLE) [9] could be used to cluster the samples and define outliers. The outliers label could be considered noise and flipped. Data set could be alternatively relabeled with the snorkel [4] Python library. The samples which labels were flipped should be compared to the samples found with cleanlab or OTRIMLE. If all models mark the same samples as noise, noise correction may be validated and used in further classification methods.

The implemented models could be integrated into production to assess their performance on the new real data. The binary classification could be transformed into the probabilistic output, and the required certainty could be empirically adjusted in the production. In addition, the classifiers could be combined into ensembles to increase the overall accuracy. More experiments could be performed with data from the second test, although this would require greater testing time.

As was shown in the results of the experiments in Chapter 4, the data cleaning seems to be the main stumbling block for the highly accurate model. Rather than applying existing classification strategies, future research should focus on data quality and failure root cause representation. The supervised or unsupervised methods for the root cause research could also boost the overall performance of the model.

Appendix A

Further Information

List of Figures

Bibliography

[1] *Rapidminer*. https://rapidminer.com/. Accessed: 2020-12-09.

[2] *Rapidminer Documentaition: Operator Manual*. https://docs.rapidminer.com/latest/studio/operators/. Accessed: 2020-12-09.

[3] *sklearn Documentaition: Random Forest*. https://scikit-learn.org/stable/modules/ensemble.html#forest. Accessed: 2020-12-09.

[4] *snorkel library*. https://github.com/snorkel-team/snorkel. Accessed: 2020-12-09.

[5] Abraham, Ranjit, Jay Simha and Sheena Iyengar: *A comparative analysis of discretization methods for Medical Datamining with Naïve Bayesian classifier*. pages 235–236, 12 2006.

[6] Argyro Kampouraki, George Manis and Christophoros Nikou: *Heartbeat Time Series Classification With Support Vector Machines*. IEEE Transactions on Information Technology in Biomedicine, 13(4):512–518, 2009.

[7] Breiman, Leo: *Random Forests*. Machine Learning, 45:5–32, 2001.

[8] Brügge, Kai: *Analyse von IceCube-Daten und Vergleich von Voting-Mechanismen für Random Forest*. Master's thesis, Dortmund, 12 2011.

[9] Coretto, Pietro and Christian Hennig: *Robust Improper Maximum Likelihood: Tuning, Computation, and a Comparison With Other Methods for Robust Gaussian Clustering*. Journal of the American Statistical Association, 111(516):1648–1659, Oct 2016.

[10] Fang, Yongsheng and Jun li: *A Review of Tournament Selection in Genetic Programming*. pages 181–192, 10 2010.

[11] Frenay, B. and M. Verleysen: *Classification in the Presence of Label Noise: A Survey*. IEEE Transactions on Neural Networks and Learning Systems, 25(5):845–869, 2014.

[12] Gärtner, Dustin: *Klassifikation von Zeitreihen über die Bestimmung häufiger symbolisierter Subsequenzen.* Master's thesis, Dortmund, 12 2014.

[13] Hand, David and Keming Yu: *Idiot's Bayes: Not So Stupid after All?* International Statistical Review, 69:385 – 398, 05 2007.

[14] Le Nguyen, Thach, Severin Gsponer, Iulia Ilie and Georgiana Ifrim: *Interpretable Time Series Classification using All-Subsequence Learning and Symbolic Representations in Time and Frequency Domains*, 08 2018.

[15] Liaw, Matthew Wiener Andy: *Classification and regression by randomforest.* 2002.

[16] Lin, Jessica, Eamonn Keogh, Stefano Lonardi and Bill Chiu: *A Symbolic Representation of Time Series, with Implications for Streaming Algorithms.* pages 2–11, 01 2003.

[17] Matuschek, Christian: *Symbolisierung und Clustering von Zeitreihen als neue Operatoren im ValueSeries Plugin von Rapidminer.* Master's thesis, Dortmund, 09 2013.

[18] Mierswa, Ingo: *Evolutionary Algorithms for Feature Selection: Part 2.* `https://rapidminer.com/blog/evolutionary-algorithms-feature-selection/`. Accessed: 2020-12-09.

[19] Mierswa, Ingo and Katharina Morik: *Automatic Feature Extraction for Classifying Audio Data.* Machine Learning, 58:127–149, 02 2005.

[20] Northcutt, Curtis, Lu Jiang and Isaac Chuang: *Confident Learning: Estimating Uncertainty in Dataset Labels*, 10 2019.

[21] R. LEARDI, R. BOGGIA and M. TERRILE: *GENETIC ALGORITHMS AS A STRATEGY FOR FEATURE SELECTION.* JOURNAL OF CHEMOMETRICS, 6:267–281, 1992.

[22] Reis, Itamar, Dalya Baron and Sahar Shahaf: *Probabilistic Random Forest: A machine learning algorithm for noisy datasets*, 11 2018.

[23] Rodríguez, Juan and Ludmila Kuncheva: *Time Series Classification: Decision Forests and SVM on Interval and DTW Features.* Proc Workshop on Time Series Classification, 13th International Conference on Knowledge Discovery and Data mining, 2007.

[24] Schäfer, Patrick and Mikael Högqvist: *SFA: A symbolic fourier approximation and index for similarity search in high dimensional datasets.* pages 516 – 527, 03 2012.

[25] Stolpe, Marco, Hendrik Blom and Katharina Morik: *Sustainable Industrial Processes by Embedded Real-Time Quality Prediction*, pages 201–243. 04 2016.

[26] Toloşi, Laura and Thomas Lengauer: *Classification with correlated features: unreliability of feature ranking and solutions.* Bioinformatics, 27(14):1986–1994, 05 2011.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kentlich gemacht habe.

Dortmund, den May 3, 2021

Olga Akulinushkina