Procedia CIRP

Predictive analytics in quality assurance for assembly processes: lessons learned from a case study at an industry 4.0 demonstration cell

--Manuscript Draft--

Manuscript Number:	PROCIR-D-20-00532R1
Article Type:	SI: CMS 2021
Section/Category:	SI: CMS 2021
Corresponding Author:	Alejandro Ricardo Perez Martinez Universität Siegen: Universitat Siegen Siegen, Deutschland GERMANY
First Author:	Peter Burggräf
Order of Authors:	Peter Burggräf
	Johannes Wagner
	Benjamin Heinbach
	Fabian Steinberg
	Alejandro Ricardo Perez Martinez
	Lennart Schmallenbach
	Jochen Garcke
	Daniela Steffes-Lai
	Moritz Wolter
Abstract:	Quality assurance (QA) is an important task in manufacturing to assess whether products meet their specifications. However, QA might be expensive, time-consuming, or incomplete. This paper presents a solution for predictive analytics in QA based on machine sensor values during production while employing specialized machine-learning models for classification in a controlled environment. Furthermore, we present lessons learned while implementing this model, which helps to reduce complexity in further industrial applications. The paper's outcome proves that the developed model was able to predict product quality, as well as to identify the correlation between machine-status and faulty product occurrence.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

```
This is pdfTeX, Version 3.14159265-2.6-1.40.21 (TeX Live 2020/W32TeX)
(preloaded format=pdflatex 2020.5.12)  15 DEC 2020 07:28
entering extended mode
 restricted \write18 enabled.
 %&-line parsing enabled.
**paper.tex
(./paper.tex
LaTeX2e <2020-02-02> patch level 5
L3 programming layer <2020-05-05> (c:/TeXLive/texmf-
local/tex/latex/aries/elsar
ticle.cls
Document Class: elsarticle 2008/10/09, 1.0.2: Elsevier Science
\@bls=\dimen134
(c:/TeXLive/2020/texmf-dist/tex/latex/base/article.cls
Document Class: article 2019/12/20 v1.4l Standard LaTeX document class
(c:/TeXLive/2020/texmf-dist/tex/latex/base/size10.clo
File: size10.clo 2019/12/20 v1.4l Standard LaTeX file (size option)
)
\c@part=\count167
\c@section=\count168
\c@subsection=\count169
\c@subsubsection=\count170
\c@paragraph=\count171
\c@subparagraph=\count172
\c@figure=\count173
\c@table=\count174
\abovecaptionskip=\skip47
\belowcaptionskip=\skip48
\bibindent=\dimen135
) (c:/TeXLive/2020/texmf-dist/tex/latex/graphics/graphicx.sty
Package: graphicx 2019/11/30 v1.2a Enhanced LaTeX Graphics (DPC,SPQR)
(c:/TeXLive/2020/texmf-dist/tex/latex/graphics/keyval.sty
Package: keyval 2014/10/28 v1.15 key=value parser (DPC)
\KV@toks@=\toks15
) (c:/TeXLive/2020/texmf-dist/tex/latex/graphics/graphics.sty
Package: graphics 2019/11/30 v1.4a Standard LaTeX Graphics (DPC,SPQR)
(c:/TeXLive/2020/texmf-dist/tex/latex/graphics/trig.sty
Package: trig 2016/01/03 v1.10 sin cos tan (DPC)
) (c:/TeXLive/2020/texmf-dist/tex/latex/graphics-cfg/graphics.cfg
File: graphics.cfg 2016/06/04 v1.11 sample graphics configuration
)
Package graphics Info: Driver file: pdftex.def on input line 105.
(c:/TeXLive/2020/texmf-dist/tex/latex/graphics-def/pdftex.def
File: pdftex.def 2018/01/08 v1.0l Graphics/color driver for pdftex
))
\Gin@req@height=\dimen136
\Gin@req@width=\dimen137
) (c:/TeXLive/2020/texmf-dist/tex/latex/psnfss/pifont.sty
Package: pifont 2020/03/25 PSNFSS-v9.3 Pi font support (SPQR)
LaTeX Font Info:    Trying to load font information for U+pzd on input
line 63.

(c:/TeXLive/2020/texmf-dist/tex/latex/psnfss/upzd.fd
File: upzd.fd 2001/06/04 font definitions for U/pzd.
```

```
)
LaTeX Font Info:    Trying to load font information for U+psy on input
line 64.

(c:/TeXLive/2020/texmf-dist/tex/latex/psnfss/upsy.fd
File: upsy.fd 2001/06/04 font definitions for U/psy.
))
\c@tnote=\count175
\c@fnote=\count176
\c@cnote=\count177
\c@ead=\count178
\c@author=\count179
\@eadauthor=\toks16
\c@affn=\count180
\absbox=\box45
\keybox=\box46
\Columnwidth=\dimen138
\space@left=\dimen139
\els@boxa=\box47
\els@boxb=\box48
\leftMargin=\dimen140
\@enLab=\toks17
\@sep=\skip49
\@@sep=\skip50
(./paper.spl) (c:/TeXLive/2020/texmf-dist/tex/latex/natbib/natbib.sty
Package: natbib 2010/09/13 8.31b (PWD, AO)
\bibhang=\skip51
\bibsep=\skip52
LaTeX Info: Redefining \cite on input line 694.
\c@NAT@ctr=\count181
)
\splwrite=\write3
\openout3 = `paper.spl'.

(c:/TeXLive/2020/texmf-dist/tex/latex/geometry/geometry.sty
Package: geometry 2020/01/02 v5.9 Page Geometry
(c:/TeXLive/2020/texmf-dist/tex/generic/iftex/ifvtex.sty
Package: ifvtex 2019/10/25 v1.7 ifvtex legacy package. Use iftex instead.
(c:/TeXLive/2020/texmf-dist/tex/generic/iftex/iftex.sty
Package: iftex 2020/03/06 v1.0d TeX engine tests
))
\Gm@cnth=\count182
\Gm@cntv=\count183
\c@Gm@tempcnt=\count184
\Gm@bindingoffset=\dimen141
\Gm@wd@mp=\dimen142
\Gm@odd@mp=\dimen143
\Gm@even@mp=\dimen144
\Gm@layoutwidth=\dimen145
\Gm@layoutheight=\dimen146
\Gm@layouthoffset=\dimen147
\Gm@layoutvoffset=\dimen148
\Gm@dimlist=\toks18
) (c:/TeXLive/2020/texmf-dist/tex/latex/base/fleqn.clo
```

```
File: fleqn.clo 2016/12/29 v1.2b Standard LaTeX option (flush left
equations)
\mathindent=\dimen149
Applying: [2015/01/01] Make \[ robust on input line 50.
LaTeX Info: Redefining \[ on input line 51.
Already applied: [0000/00/00] Make \[ robust on input line 62.
Applying: [2015/01/01] Make \] robust on input line 74.
LaTeX Info: Redefining \] on input line 75.
Already applied: [0000/00/00] Make \] robust on input line 83.
)) (c:/TeXLive/2020/texmf-dist/tex/latex/txfonts/txfonts.sty
Package: txfonts 2008/01/22 v3.2.1
LaTeX Font Info:    Redeclaring symbol font `operators' on input line 21.
LaTeX Font Info:    Overwriting symbol font `operators' in version
`normal'
(Font)                  OT1/cmr/m/n --> OT1/txr/m/n on input line 21.
LaTeX Font Info:    Overwriting symbol font `operators' in version `bold'
(Font)                  OT1/cmr/bx/n --> OT1/txr/m/n on input line 21.
LaTeX Font Info:    Overwriting symbol font `operators' in version `bold'
(Font)                  OT1/txr/m/n --> OT1/txr/bx/n on input line 22.
\symitalic=\mathgroup4
LaTeX Font Info:    Overwriting symbol font `italic' in version `bold'
(Font)                  OT1/txr/m/it --> OT1/txr/bx/it on input line 26.
LaTeX Font Info:    Redeclaring math alphabet \mathbf on input line 29.
LaTeX Font Info:    Overwriting math alphabet `\mathbf' in version
`normal'
(Font)                  OT1/cmr/bx/n --> OT1/txr/bx/n on input line 29.
LaTeX Font Info:    Overwriting math alphabet `\mathbf' in version `bold'
(Font)                  OT1/cmr/bx/n --> OT1/txr/bx/n on input line 29.
LaTeX Font Info:    Redeclaring math alphabet \mathit on input line 30.
LaTeX Font Info:    Overwriting math alphabet `\mathit' in version
`normal'
(Font)                  OT1/cmr/m/it --> OT1/txr/m/it on input line 30.
LaTeX Font Info:    Overwriting math alphabet `\mathit' in version `bold'
(Font)                  OT1/cmr/bx/it --> OT1/txr/m/it on input line 30.
LaTeX Font Info:    Overwriting math alphabet `\mathit' in version `bold'
(Font)                  OT1/txr/m/it --> OT1/txr/bx/it on input line 31.
LaTeX Font Info:    Redeclaring math alphabet \mathsf on input line 40.
LaTeX Font Info:    Overwriting math alphabet `\mathsf' in version
`normal'
(Font)                  OT1/cmss/m/n --> OT1/txss/m/n on input line 40.
LaTeX Font Info:    Overwriting math alphabet `\mathsf' in version `bold'
(Font)                  OT1/cmss/bx/n --> OT1/txss/m/n on input line 40.
LaTeX Font Info:    Overwriting math alphabet `\mathsf' in version `bold'
(Font)                  OT1/txss/m/n --> OT1/txss/b/n on input line 41.
LaTeX Font Info:    Redeclaring math alphabet \mathtt on input line 50.
LaTeX Font Info:    Overwriting math alphabet `\mathtt' in version
`normal'
(Font)                  OT1/cmtt/m/n --> OT1/txtt/m/n on input line 50.
LaTeX Font Info:    Overwriting math alphabet `\mathtt' in version `bold'
(Font)                  OT1/cmtt/m/n --> OT1/txtt/m/n on input line 50.
LaTeX Font Info:    Overwriting math alphabet `\mathtt' in version `bold'
(Font)                  OT1/txtt/m/n --> OT1/txtt/b/n on input line 51.
LaTeX Font Info:    Redeclaring symbol font `letters' on input line 58.
LaTeX Font Info:    Overwriting symbol font `letters' in version `normal'
```

```
(Font)                  OML/cmm/m/it --> OML/txmi/m/it on input line 58.
LaTeX Font Info:    Overwriting symbol font `letters' in version `bold'
(Font)                  OML/cmm/b/it --> OML/txmi/m/it on input line 58.
LaTeX Font Info:    Overwriting symbol font `letters' in version `bold'
(Font)                  OML/txmi/m/it --> OML/txmi/bx/it on input line
59.
\symlettersA=\mathgroup5
LaTeX Font Info:    Overwriting symbol font `lettersA' in version `bold'
(Font)                  U/txmia/m/it --> U/txmia/bx/it on input line 67.
LaTeX Font Info:    Redeclaring symbol font `symbols' on input line 77.
LaTeX Font Info:    Overwriting symbol font `symbols' in version `normal'
(Font)                  OMS/cmsy/m/n --> OMS/txsy/m/n on input line 77.
LaTeX Font Info:    Overwriting symbol font `symbols' in version `bold'
(Font)                  OMS/cmsy/b/n --> OMS/txsy/m/n on input line 77.
LaTeX Font Info:    Overwriting symbol font `symbols' in version `bold'
(Font)                  OMS/txsy/m/n --> OMS/txsy/bx/n on input line 78.
\symAMSa=\mathgroup6
LaTeX Font Info:    Overwriting symbol font `AMSa' in version `bold'
(Font)                  U/txsya/m/n --> U/txsya/bx/n on input line 94.
\symAMSb=\mathgroup7
LaTeX Font Info:    Overwriting symbol font `AMSb' in version `bold'
(Font)                  U/txsyb/m/n --> U/txsyb/bx/n on input line 103.
\symsymbolsC=\mathgroup8
LaTeX Font Info:    Overwriting symbol font `symbolsC' in version `bold'
(Font)                  U/txsyc/m/n --> U/txsyc/bx/n on input line 113.
LaTeX Font Info:    Redeclaring symbol font `largesymbols' on input line
120.
LaTeX Font Info:    Overwriting symbol font `largesymbols' in version
`normal'
(Font)                  OMX/cmex/m/n --> OMX/txex/m/n on input line 120.
LaTeX Font Info:    Overwriting symbol font `largesymbols' in version
`bold'
(Font)                  OMX/cmex/m/n --> OMX/txex/m/n on input line 120.
LaTeX Font Info:    Overwriting symbol font `largesymbols' in version
`bold'
(Font)                  OMX/txex/m/n --> OMX/txex/bx/n on input line 121.
\symlargesymbolsA=\mathgroup9
LaTeX Font Info:    Overwriting symbol font `largesymbolsA' in version
`bold'
(Font)                  U/txexa/m/n --> U/txexa/bx/n on input line 129.
LaTeX Font Info:    Redeclaring math symbol \mathsterling on input line
164.
LaTeX Font Info:    Redeclaring math symbol \hbar on input line 591.
LaTeX Info: Redefining \not on input line 1043.
)

! LaTeX Error: File `ecrc.sty' not found.

Type X to quit or <RETURN> to proceed,
or enter new name. (Default extension: sty)

Enter file name:
! Emergency stop.
<read *>
```

```
l.40 \usepackage
                {amsmath}^^M
*** (cannot \read from terminal in nonstop modes)


Here is how much of TeX's memory you used:
 2743 strings out of 480681
 32136 string characters out of 5908536
 267131 words of memory out of 5000000
 18596 multiletter control sequences out of 15000+600000
 533028 words of font info for 26 fonts, out of 8000000 for 9000
 1141 hyphenation exceptions out of 8191
 38i,0n,34p,212b,86s stack positions out of
5000i,500n,10000p,200000b,80000s
!  ==> Fatal error occurred, no output PDF file produced!
```

Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 00 (2021) 000–000

www.elsevier.com/locate/procedia

54th CIRP Conference on Manufacturing Systems

Predictive analytics in quality assurance for assembly processes: lessons learned from a case study at an industry 4.0 demonstration cell

Peter Burggräf[a], Johannes Wagner[a], Benjamin Heinbach[a], Fabian Steinberg[a], Alejandro R. Pérez M.*[a], Lennart Schmallenbach[a], Jochen Garcke[b,c,d], Daniela Steffes-lai[b], Moritz Wolter[b,c,e]

[a] *Chair of International Production Engineering and Management (IPEM), Universität Siegen, Paul-Bonatz-Straße 9-11, 57076 Siegen, Germany*
[b] *Fraunhofer Institute for Algorithms and Scientific Computing (SCAI), Schloss Birlinghoven 1, 53757 Sankt Augustin, Germany*
[c] *Fraunhofer Center for Machine Learning, Schloss Birlinghoven 1, 53757 Sankt Augustin, Germany*
[d] *Institute for Numerical Simulation, Universität Bonn, Endenicher Allee 19b, 53115 Bonn, Germany*
[e] *Institute for Computer Science, Universität Bonn, Endenicher Allee 19a, 53115 Bonn, Germany*

* Corresponding author. Tel.: +49-271-740-2630; fax: +49-271-740-12628. *E-mail address:* alejandro.perez@uni-siegen.de

Abstract

Quality assurance (QA) is an important task in manufacturing to assess whether products meet their specifications. However, QA might be expensive, time-consuming, or incomplete. This paper presents a solution for predictive analytics in QA based on machine sensor values during production while employing specialized machine-learning models for classification in a controlled environment. Furthermore, we present lessons learned while implementing this model, which helps to reduce complexity in further industrial applications. The paper's outcome proves that the developed model was able to predict product quality, as well as to identify the correlation between machine-status and faulty product occurrence.

Keywords: Machine-Learning; Predictive Quality; Production; Quality Assurance; Classification

1. Introduction

For companies aiming at quality leadership, the final quality of any goods produced is vital for market success. Thus, companies need to ensure that every product meets the expected customer and legal requirements. The quality control (QC) of production processes and products is one fundamental part of general quality management. It contains the measures and activities needed to fulfill specific quality requirements [14].

Over the years, diverse methods to ensure the quality of products have been developed [14]. Among the most used methods, we have Auditing, Total Quality Control (TCQ) [10], Poka-Yoke Techniques [17], the 100% control of each produced product [9], and statistical methods like the Statistical Process Control (SPC) [19]. Independently of the method used, the goal is the same, to avoid the production of faulty products, or to identify and remove them, before delivering those to the customer.[14]. However, the relationship between cost and efficiency varies from one method to another. This is noticeable, e.g., at the cost of highly specialized equipment for automated inspection, or the cost of employees performing a manual inspection of the produced products versus the accuracy of detecting all faulty products [18].

In general, the quality characteristics of a product can be classified as attributes or variables [14]. *Attributes* are qualitative measurable characteristics, i.e., color or texture. On the other hand, *variables* are precise and measurable characteristics, which are shown as numbers, i.e., length or width. The advantage of quantitative measurements is that in the event of a defective intermediate or end product, precise measured values are available, which enable controlled adjustment of the process parameters, as well as the possibility of observing trends in the collected data. This enables the employment of predictive models to determine the quality of the product before the QC process takes place [3, 11].

With the evolution of machine learning (ML) applications [3, 4], approaches combining QC and predictive models are becoming more relevant. The applications range from the defect analysis of the produced piece [6, 20], to process-oriented QC [12, 11, 2, 16], see Section 2. Independently of the research

area previously stated, within the scope of the related works, there is no evidence of an evaluation framework for diverse ML techniques applied in QC process.

Considering the high amount of data available in industries, one approach to reduce the inspection costs and simultaneously increase the accuracy of detecting faulty products would be to implement an ML model capable to predict the product quality based on the initial state of the machine. This will not only help to detect faulty products before the QC process is performed, it would also strengthen the current QC methods by combining them with QC predictive models.

While implementing ML models to QC, several challenges arise that have not yet been sufficiently addressed, e.g., what to consider while evaluating the output from ML models for industrial applications? (model accuracy vs applicability at industrial environments); what steps should be followed to prepare the data for predictive QC methods?; and what findings and lessons are worth to consider for further applications in the area?. For this, we cover the methods used for quality checks in production, as well as the bases of predictive analytics in production (Section 2). We describe a suitable industrial platform to implement the ML models (Section 3). Then, we explain the three machine learning methods implemented in this work: Multilayer perceptrons, Support Vector Machines, and Decision Tree (Section 4), followed by the description of the experiment process from the data acquisition and preprocessing, to the classifier optimization and testing (Section 5). Finally, we present the findings achieved after the finalization of this project phase (Section 6), as well as present a list of lessons learned for the implementation of the project, which might be helpful for further implementations (Section 7).

2. Related Works

According to Delen and Damirkan, predictive analytics (PA) consists of unraveling the inherent relationships (if any) between input and output by using data and mathematical techniques [5]. PA offers a wide range of applications, and it can be implemented, as long as sufficient data are available [3]. According to Krauß, there is a wide range of applications for PA in the industrial and production environment. Some common industrial applications of PA are focussed in products (design and optimization), machines and assets (predictive maintenance, anomaly detection, self learning-machines) and process (scheduling, process design, predictive process control) [11].

An explorative literature review on this matter reveals that publications that combine PA and QC became more relevant in the last six years. This could be due to computer power limitations in earlier research and the recent increase of ML applications in this area. Related works mostly belongs to two groups, defect analysis of produced pieces and process-oriented QC. Due to the recent advances in ML approaches specialized in image and pattern recognition [3], a great number of publications focus on product-analysis based on images from cameras during the production process, e.g., Escobar employed pattern recognition for defect detection in a binary classification problem based on a l_1-regularized logistic regression [6]; Sohnius focussed on defect prediction within printed layers by employing supervised models based on features extracted directly from the machine code and the scan of the layers surface [20]. Other authors focussed their research on process-oriented QC, e.g., Krauß focussed on describing the ML pipeline to implement automated machine learning for predictive quality in production [12] and product quality prediction in a process chain [11]; Aumi focussed on the model development for predictive quality control of batch processes [2]; Ritter studied the ways of process modeling on quality prediction and assurance of chipboards [16].

Applications that combine PA with QC might improve the current QC processes. However, based on the previous findings, there are no clear references to the impact of implementing ML models based just on model accuracy, together with industrial factors of the QC process. By employing machine learning (ML) techniques (Section 4), together with the QC traditional methods, it would be possible to improve the relationship between inspection costs and the probability of detecting faulty products during the QC process, as well as, decrease the risk of labeling NOK pieces as OK and deliver them to costumers. However, before implementing these techniques, it is important to define the platform where these predictive analytics models will be implemented in a controlled and transparent industrial environment.

3. The case: industry 4.0 demonstration cell

Our case study is an abstraction of an assembly process of the industry. This case study is a highly flexible and transparent platform regarding software and hardware access and manipulation. The industry 4.0 demonstration cell is composed of three independent conveyor belts, a robotic assembly arm (UR3), a laser scanner used for quality control as well as a wide range of sensors, all orchestrated by a SIEMENS PLC S7-1200, see Figure 1. The executed abstracted assembly process consists of stacking two disks of different sizes on top of each other. The corresponding pseudo product quality is later determined using the laser scanner and evaluating the concentricity of both disks. If the disks' concentricity is within a tolerance of 1.5mm, the piece is classified as OK, if not, it is classified as NOK. By considering that malfunctions can occur during industrial production processes, our demonstration cell can simulate failures in diverse areas, such as:

- Assembly errors due to the robotic arm: incorrectly positioned disks.
- Bearing damage on the conveyor belts: vibration changes in the conveyor belt motors.
- Resistance on the conveyor belts: temperature changes in the conveyor belt motors.
- Malfunction of the gate door due to leakage in the compressed air system: cylinder stroke at the assembly barrier is slower.

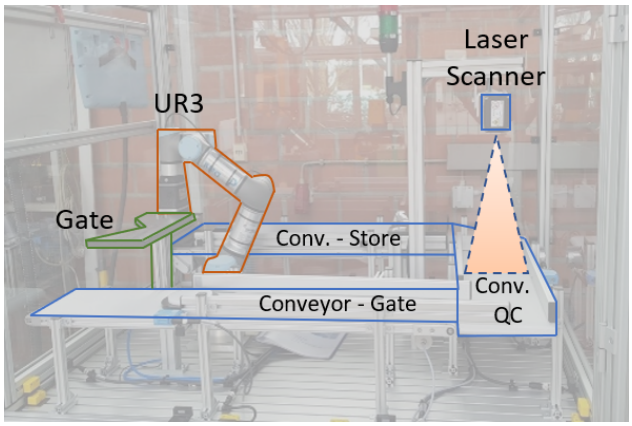

Fig. 1: The industry 4.0 demonstration cell at the University of Siegen

- Faster operating point: disks shift position due to the high belt speeds.
- Missing material at the end of the warehouse: production stop due to lack of material.

Those simulated failures have an impact on the data collected by the sensors. Anomalies can be observed at the vibration sensors, temperature rise, fluctuations in the air pressure system, noticeable shifting on pieces' position as well as the stop of the process.

The continuous data collection and the clear correlation between the machine parameters and the quality of the product, is the ideal play-ground to implement a sandbox for machine learning implementations. With this in mind, our goal is to predict the quality of the product (concentricity of the assembled parts), before the actual quality check done by the laser scanner.

4. Machine Learning Methods

To study which implementation would be more suitable for industrial applications, we compare Multilayer Perceptrons (MLPs), Support Vector Machines (SVM), as well as Decision Trees on the task of quality prediction. All three will be introduced next.

4.1. Multilayer perceptrons

These feedforward neural networks typically combine multiple layers and activation functions [3]. A layer contains a large weight matrix and a bias vector. To evaluate the layer the input vector must be multiplied with the weight matrix before the bias vector is added. Each layer is typically followed by an activation function which adds non-linearity to the network graph. Rectified linear units (ReLUs) are zero for negative inputs and leave positive inputs unchanged. ReLUs are the recommended activation function for modern neural networks [8], we use these here as well. Since the linear parts of the ReLU and the linear matrix multiplication are differentiable we can employ stochastic gradient descent to train our classifier.

4.2. Support Vector Machines

Support Vector Machines (SVM) attempt to separate data along hyperplanes [1]. SVM offer a convex alternative to MLPs [21]. A convex classifier is guaranteed to converge to a global solution regardless of its initialization. Non-separable data can become separable if projected to a different possibly high dimensional space through a Kernel function. The Kernel trick allows SVM to solve some non-linear classification problems [21]. Radial basis functions (RBF) are commonly chosen as kernels [21], therefore, we choose to follow this practice as well.

4.3. Decision Tree

Decision Trees are essentially binary trees, which work with the input features at the roots, decisions are made by moving up the tree to top leaves. At every branch in the tree, a decision is made, until one arrives at the decision [13]. For numerical input features, each decision partitions the data [1], for example by comparison to a threshold value. To construct the tree these cutoff values have to be chosen at every branch. The construction problem turns into an optimization problem when a split criterion is introduced. Common choices are cross-entropy of Gini-coefficient [1]. During the construction splits minimizing, the selected functions are chosen.

5. Experiments

5.1. Data acquisition

Industrial projects require to work with a variety of sensors and controllers from different manufacturers. Thus, the risk of incompatibility between technologies increases. To avoid these problems, universal protocols are often implemented in industrial projects. Figure 2 describes a simplified version of the real connection diagram of the demonstration cell. In our case, the data is acquired independently from two OPC-UA servers. This represents the interaction between multiple servers in real production environments.

Before collecting the data, we needed to find the variables of interest and map their source, as well as to determine the use cases of interest. Our study focussed on the data described in Table 1 and the six use cases described in Table 2. The data was manually collected by using the software "UAExpert - v1.5.1". Approximately 15 hours of data were collected in different sessions, while equally sampling each use case during the data acquisition stage.

5.2. Data preprocessing

Once the data was collected, the next step was to clean the raw data and organize it accordingly [3]. In our case, that means to merge the data collected from the two OPC-UA servers, and synchronize the time-stamp. Since our systems were not completely synchronized, as common in practice, it was needed to

Data of interest	Server Source	Variable
Conveyor speed	Siemens PLC	- Conveyor: Gate - Conveyor: QC - Conveyor: Store
UR3 Position	Siemens PLC	- *xyz* Gripper position
Action Flags	Siemens PLC	- UR3 gripper open/close - Grab/drop disk based on type
Quality control	SICK SIM4000	- OK-NOK label - *xy* disk deviation from the center point - Absolute disk deviation
Gate position	SICK SIM4000	- Position - Speed - Safe to go
Store	SICK SIM4000	- Disk Size

Table 1: Industry 4.0 demonstration cell: Relevant data linked with its reference source server.

Use Case	Description
Use Case 1	Conveyor speed: Slow Robot position: OK
Use Case 2	Conveyor speed: Slow Robot position: NOK
Use Case 3	Conveyor speed: Fast Robot position: OK
Use Case 4	Conveyor speed: Fast Robot position: NOK
Use Case 5	Conveyor speed: Too Fast Robot position: OK
Use Case 6	Conveyor speed: Too Fast Robot position: NOK

Table 2: Industry 4.0 demostration cell: Controlled Use-Cases. Use-Cases with the value *Robot position: NOK* and *Conveyor speed: Too Fast* will result in NOK pieces.

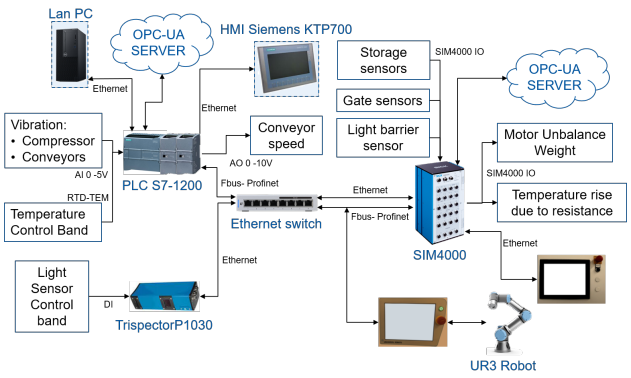

Fig. 2: Industry 4.0 demostration cell: Connection diagram (simplified).

calculate the delta-time between servers by referencing each server's base clock. Once the delta-time was determined, a time-stamp function was implemented to correct the time-shift between samples and be able to merge all the data without incompatibilities.

Data tags interpretation is an important step in the preprocessing process [3]. The raw data collected from the sensors is usually tagged by an alphanumeric code predefined by the sensor manufacturer. To work with the data, it was needed to implement diverse functions which have the task to clean the raw data in a more human-readable form. This means to remove samples out of clear boundaries and missing values, as well as to translate the sensor code tags based on human-readable words predefined by the authors. This last step, might be optional in other implementations, however, it proves to be a great addition for testing purposes.

Once the data was cleaned and compiled in one directory, the next step was to split the data based on the assembly cycle. For our purpose, we split the dataset into individual data-samples with help of the *action flags*included in the dataset Table 1. Subsequently, each data-sample was automatically checked for missing sensor values and discarded if so.

Each time-series sample was the result of data collected during one machine assembly cycle (MAC). The starting point of one MAC was defined as the very first moment the robotic arm grabs the white disk from the store area and the ending point was defined as the starting of the subsequent MAC, the grabbing of the next white disk in the same area. To eliminate time as a variable in our dataset, it was needed to transform time-series into feature data suitable describing a MAC in a way that different MACs are easily comparable. For that reason, we defined a set of sampling-conditions that helped us to collect the relevant data during the MAC:

- Value of the robot position (*xyz*) while dropping both disks in the *conveyor - gate*
- Mean conveyor speed value per conveyor band
- Absolute deviation (*xy*) value at the quality control check
- Absolute quality control value: OK - NOK

Each time-series sample was evaluated based this sampling-condition, and the result of each evaluation was considered as our atemporal data-sample, which we used as the base for our classifier.

5.3. Classifier optimization and testing

The robot's position at disk drop is marked in Figure 3. The quality prediction problem was framed as a classification task. The input vectors which we fed into our classifiers consist of the arm positions at the disk drops for both disks in three dimensions, as well as the largest recorded belt speed of all three belts. The interpretation of the quality measurement was used as the training target. It can be OK or NOK which we encoded as zero and one.

We worked with a total of 528 measurements. Each containing arm and belt data logs of individual cell run. 50 samples were set aside at random for testing purposes, leaving 478 training samples. The random number generator seed was set to one

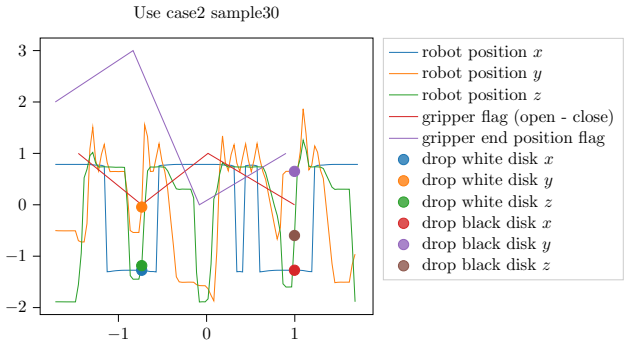

Fig. 3: Robot arm movement patterns. We show the movement of the cell's robot arm, its grippers, and the encoding of the overall position in the system. The changes in the grip flag are triggered, when the gripper opens or closes. 0 indicates an open, while one indicates a closed gripper. The position flags mark the state of the cell's arm. 0 means the black disk is in store, one that the black disk is at the gate, two that the white disk is in store and three that the white disk is located at the gate.

Approach	accuracy
Naïve-Baseline	64 %
Support Vector Machine	82 %
Multilayer Perceptron	88 %
Decision Tree	98 %

Table 3: Comparison of Support Vector Machine (SVM), Multilayer Perceptron and Decision Tree classification on our anomaly detection task. The first row shows the performance of naively predicting a broken piece every time.

to ensure the train and test set splits are identical for all experiments.

We compared a total of three different classifier architectures on the data, a Support Vector Machine (SVM), a Multilayer Perceptron (MLP) and a Decision Tree structure. In this case, we chose to train all three models using exemplary default hyper-parameters, which we found to work well. [1]

Results are shown in table 3. For 32 of the total 50 test samples quality measurements indicated a problem. This set the baseline over the entire data set, which would be obtained by simply labeling all samples as faulty. For the test set, we required to classify more than 64% of the data correctly. Therefore, we expected any naive classifier to produce at least 64% accuracy, which we had to surpass. In Table 3, we observe that this was indeed the case for all three approaches evaluated here.

The MLP prediction results presented on Table 4b show that even though the prediction model was not 100% accurate, the percentage of false-positive predictions was 0%. The false-negative results represented 12% of the test sample and the true predictions constituted 88% of the test data.

The outcome of the models showed that the Decision Tree performed best followed by the Multilayer Perceptron and the Support Vector Machine. However, by reviewing the confusion matrices presented in Table 4, Multilayer Perceptron would be

[1] To allow exact reproduction of our results source code is available at https://github.com/manubrain/Demo-Cell-Classification

True Values

Predicted Values	OK	NOK	Total
OK	18	0	18 + 0 = 18
NOK	8	24	8 + 24 = 32
Total	18 + 8 = 26	0 + 24 = 24	50

(a) Support Vector Machine.

True Values

Predicted Values	OK	NOK	Total
OK	18	0	18 + 0 = 18
NOK	6	26	6 + 26 = 32
Total	18 + 6 = 24	0 + 26 = 26	50

(b) Multilayer Perceptron.

True Values

Predicted Values	OK	NOK	Total
OK	17	1	17 + 1 = 18
NOK	0	32	0 + 32 = 32
Total	17 + 0 = 17	1 + 32 = 33	50

(c) Decision Tree.

Table 4: Confusion matrices result of the prediction of 50 test data-points. We present the result of the predictions given by all three algorithms in detail based on the comparison between true values vs predictions. The tables read from the upper-left cell as: true-positive, false-positive, false-negative, true-negative.

the preferred solution to be implemented in industrial applications due to the 0% ratio in false-positive results and high accuracy of 88%.

6. Conclusion

Based on the explorative literature review, we observed that the hundred percent quality check method proves to be the most efficient method to guarantee the final product quality, however, the high implementation cost is its biggest draw-back. On the other hand, we have the statistical methods, which are known to be fast and efficient, but the lack of reviewing each product until the next batch review might imply a considerable loss of the produced products whenever NOK products are found. This is based on the uncertainty of when the error occurred and how many products are affected.

Implementing ML models based on just the initial conditions of the process and capable to predict the quality of the product, would be a great asset for the industrial processes. Within the limits of our work, we predicted with 98% certainty the product quality by implementing a Decision Tree model with a 6-variables vector as input. However, our decision for industrial applications would be the MLP with 88% accuracy, to guarantee that all passed parts are OK and avoid labeling NOK pieces as OK. For further work, we expect to increase the accuracy of the prediction model by collecting more data-samples and experimenting with other, more complex, ML models.

It is important to state that this prediction model is not yet ready to fully replace the statistical methods, nor the hundred percent quality check method, since this research is still in its early stages and needs further development. However, a combination of SPC along prediction models could work well together, since the prediction model will evaluate each produced

piece, and this might compensate for disadvantages of the statistical methods.

7. Lesson Learned

It is worth to mention the most important points learned during the project implementation. This might help as building blocks for further research in the area. Difficulties at implementing ML models in production are also described by others authors [15].

Incompatibility between technologies; devices configured on the same network presented troubles to share data due to network restrictions, as well as the limitation to access and modify restricted program-code given by the machine manufacture. To overcome the situation stated above, it was necessary to reprogram big sections of the machine control system.

Data synchronization; two OPC-UA servers were used on this implementation. This led to a series of data synchronization problems due to differences in the internal clock of both servers. To solve the issue, it was needed to write a time-stamp synchronization script that was deployed during the data preprocessing phase. Without harmonious time-stamps, determining the production cycle would have been impossible and the input data for the ML model would have been useless, and the model inaccurate.

Data interpretation; this process is considered complicated, time-consuming, and a fundamental step for any ML applications. It is fundamental to understand the assembly process, the timing from the machine movements, the boundaries of the machine's sensors, as well a basic correlation within the presented data.

Missing values; to minimize the risk of false results delivered by the ML model, it would be compulsory to carefully review the ML input data. Missing values can have a great impact on the ML prediction, that is the reason why we deliberately review all the input data and validate it with the expected output from the diverse sensors. Whenever there was missing data, and as long as we did not fabricate data, we employed statistical methods to fill the data gap.

Adapting the industry 4.0 demonstration cell; the initial state of the demonstrator was very limited. Even though the process was well known, the lack of labels or flags whenever certain actions occur, made it complicated to have accurate data sampling from each assembly cycle. To overcome this uncertainty, we modified the machine program to raise specific flags whenever certain actions were executed.

Machine Learning; careful measurement and meaningful preprocessing of our cell-data was key to the successful application of all three algorithms. Without the labels, the problem could not have been framed as a supervised classification problem.

Dataset size; as the starting point we collected as many data points like the ones found in the iris Dataset [7]. The next step was to collect batches of data of around 30 minutes per use case. This process was repeated until accumulate around 15 hours of data.

8. Author Contributions

Pérez M. and Wolter performed writing - original draft, conceptualization, review, and editing; investigation; software and data curation. Schmallenbach performed writing - review & editing. Burggräf, Wagner, Heinbach, Steinberg performed writing - conceptualization, review. Garcke, and Steffes-lai performed writing - review.

Acknowledgments

This work was funded by the European Regional Development Fund (ERDF) within the project "ManuBrain".

References

[1] Aggarwal, C.C., 2014. Data classification. CRC PRESS.
[2] Aumi, S., Corbett, B., P., M., 2012. Model predictive quality control of batch processes. American Control Conference .
[3] Bishop, C.M., 2006. Pattern recognition and machine learning. Springer.
[4] Burggräf, P., Wagner, J., Koke, B., 2018. Artificial intelligence in production management: A review of the current state of affairs and research trends in academia, in: 2018 International Conference on Information Management and Processing (ICIMP), pp. 82–88.
[5] Delen, D., Demirkan, H., 2013. Data, information and analytics as services. Decision Support Systems 55.
[6] Escobar, C.A., Morales-Menendez, R., 2018. Machine learning techniques for quality control in high conformance manufacturing environment. Advances in Mechanical Engineering 10.
[7] Fisher, R.A., 1936. The use of multiple measurements in taxonomic problems. Annals of Eugenics 7, 179–188.
[8] Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y., 2016. Deep learning. volume 1. MIT press Cambridge.
[9] Hinckley, C.M., 1997. Defining the best quality-control systems by design and inspection. Clinical Chemistry 43, 873–879.
[10] Illés, B., Tamás, P., Dobos, P., Skapinyecz, R., 2017. New challenges for quality assurance of manufacturing processes in industry 4.0. Solid State Phenomena 261.
[11] Krauß, J., Dorißen, J., Mende, H., Frye, M., Schmitt, R., 2019. Machine learning and artificial intelligence in production: Application areas and publicly available data sets. Production at the leading edge of technology .
[12] Krauß, J., Pacheco, B.M., Zang, H.M., Schmitt, R., 2020. Automated machine learning for predictive quality in production. Procedia CIRP 93.
[13] Marsland, S., 2015. Machine Learning: An Algorithmic Perspective. 2 ed., CRC Press.
[14] Mitra, A., 2016. Fundamentals of Quality Control and Improvement. Wiley.
[15] Paleyes, A., Urma, R.G., Lawrence, N.D., 2020. Challenges in deploying machine learning: a survey of case studies , 9.
[16] Ritter, C., Schweitzer, F., 1992. Neue wege der prozeßmodellierung für die rechnergestützte qualitätsvorhersage und -sicherung in der spanplattenfertigung. Holz als Roh- und Werkstoff 50.
[17] Robinson, H., 1997. Using poka-yoke techniques for early defect detection, in: 6th International Conference on Software Testing Analysis and Review.
[18] Sandoval-Chávez, D.A., Beruvides, M.G., 1998. Using opportunity costs to determine the cost of quality: A case study in a continuous-process industry. The Engineering Economist 43, 107–124.
[19] Selvamuthu, D., Das, D., 2018. Introduction to Statistical Mehods, Design of Experiments and Statistical Quality Control. Springer.
[20] Sohnius, F., Schlegel, P., Ellerich, M., Schmitt, R.H., 2019. Data-driven prediction of surface quality in fused deposition modeling using machine learning , 473–481.
[21] Suykens, J.A., Van Gestel, T., De Brabanter, J., De Moor, B., Vandewalle, J.P., 2002. Least squares support vector machines. World scientific.