Combining Conformance Checking and Classification of XES Log Data For The Manufacturing Domain

Matthias Ehrendorfer, Juergen-Albrecht Fassmann, Juergen Mangler, Stefanie Rinderle-Ma

University of Vienna, Vienna, Austria,
matthias.ehrendorfer@gmail.com, juergen.fassmann@gmail.com

University of Vienna, Faculty of Computer Science,
Research Group Workflow Systems and Technology, Vienna, Austria,
{juergen.mangler, stefanie.rinderle-ma}@univie.ac.at

1 Introduction

Currently, data collection on the shop floor is based on individual resources such as machines, robots, and Autonomous Guided Vehicles (AGVs). There is a gap between this approach and manufacturing orchestration software that supervises the process of creating single products and controls the ressources' interactions. This creates the need to save resource-based data streams in databases, clean it, and then re-contextualize it, i.e., by connecting it to orders, batches, and single products. Looking at this data from a process-oriented analysis point of view enables new analysis prospects. This paper utilises these prospects in an experimental way by creating BPMN models for the manufacturing of two real-world products: (1) a low volume, high complexity lower-housing for a gas-turbine and (2) a high volume, low complexity, small tolerance valve lifter for a gas turbine. In contrast to the resource-based data collection, 30+ values are modeled into the BPMN models and enacted by a workflow engine, creating execution logs in the XES standard format. Conformance checks are carried out and interpreted for both scenarios and it is shown how existing classification and clustering techniques can be applied on the collected data in order to predict good and bad parts, ex-post and potentially at run-time.

The process created for the manufacturing of both parts can be divided into a number of subprocesses which correspond to different levels of the automation pyramid (given in Fig. 1). This separation allows a better overview as well as special foci in each of the subprocesses. The subprocesses are:

– The "Order Processing" process which is responsible for detecting the start of the production and spawning a subprocess for every part produced. Therefore, this process connects a number of part production processes for an order.

This work is a result of the lab course "Business Intelligence II (2018W)" and has been partially supported and funded by the Austrian Research Promotion Agency (FFG) via the "Austrian Competence Center for Digital Production" (CDP) under the contract number 854187.

- The "Part Production" process describes the process steps as well as the order in which they are executed on the level of production or measuring steps and spawns subprocesses for each of the production steps.
- The "Production" process handles the detection of individual machines starting and finishing machining and spawns subprocesses for data collection while waiting for the production step to finish.
- The "Machining" process is responsible for capturing data sent by individual machines during machining.

The "Order Processing" process corresponds to the "Plant management level" of the automation pyramid while the "Production Process" resides on the "Process control level" and the "Machining Process" can be assigned to the "Control (PLC) level". The "Part Production" process is the link between "Order Processing" and "Production" and can therefore be found between level 2 and 3 (i.e. the second and third one counting from the top) of the automation pyramid.

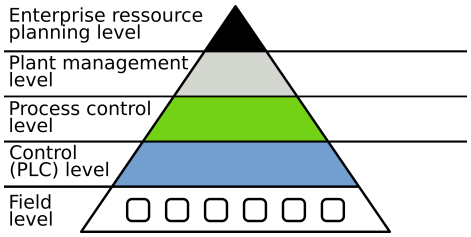


Fig. 1: Automation Pyramid

When executing the processes, log files are created. They are available as YAML files in the XES standard format and contain the tasks given in the BPMN models which are described in XML format. Production data obtained during the machining as well as measurement results are also included in the corresponding task of the log files. Due to the abovementioned usage of subprocesses, a number of log files have to be merged in order to obtain all the process steps needed for the production of one part. The information to do this (i.e. which subprocess is spawned) is also contained in the task creating the subprocess of the parent process.

As described above, the part produced in the first scenario is a low volume, high complexity lower-housing for a gas-turbine. Therefore, the parts produced are measured during the production process and repaired if they do not comply to the specifications of the plan. This leads to a higher number of machining processes as machining might be interrupted and continued a number of times. The log data for the first scenario can be download from [1].

[1] `http://cpee.org/~demo/bus_paper/data/lowerhousing.tgz`

The part produced in the second scenario is a high volume, low complexity, small tolerance valve lifter for a gas turbine. In contrast to the part produced in the first scenario it is not reasonable to repair bad parts and therefore, after the machining process manual as well as automatic measurement is performed but no repair is carried out. Due to an error while performing the manual measurement, the manual measurement results of parts 129-180 are shifted by 2. The log data for the second scenario can be downloaded from [2].

2 Conformance Checking of Shopfloor Processes

2.1 Modelling the Processes

In order to perform conformance checking, the logs as well as the templates need to be transformed to a format that allows easy checking. The ProM tool (for further information see [29]) provides the "PNetReplayer" ([2]) package which allows performance checking using the "Replay a Log on Petri Net for Conformance Analysis" function. This function takes a petri net (in a TPN format) and a XES log file which is based on XML as input files and creates a result file containing fitness values and the replay results for each log. Therefore, the BPMN process template files given as XMLs (for further information see section 1) need to be transformed to petri nets. The transformation from the BPMN process template to the petri net is performed using the following rules:

- Each "call" element is transformed into two transitions with a place in between and afterwards. The first transition is identifiable as start and the second one as end which is later used to link them to the start or complete lifecycle transitions in the logs represented as XES file.
- Each "manipulate" element is transformed into a transition with a place afterwards. As the log contains only one event for such elements there is no need to create a start and end transition.
- "terminate" elements are transformed to a transition with a place that has no outgoing edges afterwards.
- "loop" elements are transformed to two transitions (starting and closing) with a place after each of them and the second place being connected to the first transition in addition to the transition created by the element after the loop. As a loop contains other elements, the content between these two transitions is defined by the contained elements.
- "choose" elements are transformed into two places where the first one splits into a number of branches and the second one merges these branches. "alternative" and "otherwise" are both handled the same way as it does not make a difference if one of the branches is the default one when using petri nets for conformance checking. Therefore, an "alternative" or "otherwise" element is transformed into two transitions where the first one has an incoming edge from the first place created by the "choose" element and is followed by a

[2] `http://cpee.org/~demo/bus_paper/data/gv12.tgz`

place and the second one is connected to the merging place created by the "choose" element described before. As a branch contains other elements, the content of a branch is defined by the contained elements.
- "parallel" elements are transformed to two transitions. The first one is connected to a number of places equal to the number of parallel branches via edges. The second transition merges these parallel branches by having incoming edges from the last place of each parallel branch. The "parallel_branch" element is transformed to a transition signalling the start of the parallel branch with a place afterwards. As a branch contains other elements, the content of a branch is defined by the contained elements.

As a prerequisite for the transformation with the code provided, the process templates need to be in the XML format and contain only the elements defined in [7]. Additionally, the XML element "/testset/description/description" must be available and contain the elements mentioned above with the defined attributes and a valid nesting structure. Using the rules given before, the BPMN processes are then transformed to petri nets using the TPN file format.

The program implementing the rules was written in python (the code is also available at [24] for scenario 2 - at [23] for scenario 1 - and can be executed by using the following command: "./processToTpn.py [path to existing XML] [path to TPN that should be created]"):

```
#!/usr/bin/python3

import xml.etree.ElementTree as ET;
import sys;
import re;

counter=0;
termination=False;
startstack=[];
endstack=[];
loopstack=[];
printString='';

def processChild(node, indent, firstOfBranch):
  global counter;
  global startstack;
  global endstack;
  global loopstack;
  global printString;
  global termination;
  exception=False;
  indentString='';
  i=0;
  while i<indent:
    indentString+=' ';
    i+=1;
  if 'parallel_branch' in node.tag:
    print(indentString+node.tag);
    print('trans '+'x_parallel_branch');
    printString+=('trans '+'x_parallel_branch');
    printString+='\n';
    x=startstack.pop();
    print('  in '+x);
    printString+=('  in '+x);
    printString+='\n';
    counter+=1;
    print('  out p'+str(counter)+';');
```

```
38      printString+=('  out p'+str(counter)+';');
39      printString+='\n';
40      #hinzufuegen pfad zwischen transition von parallel und (neuem)
            place
41    elif 'parallel' in node.tag:
42      print(indentString+node.tag);
43      print('trans '+'x_parallel');
44      printString+=('trans '+'x_parallel');
45      printString+='\n';
46      print('  in p'+str(counter));
47      printString+=('  in p'+str(counter));
48      printString+='\n';
49      i=0;
50      outString='  out';
51      while i<len(list(node)):
52        counter+=1;
53        outString+=' p'+str(counter);
54        startstack.append('p'+str(counter));
55        if (i+1)<len(list(node)):
56          outString+=',';
57        i+=1;
58      outString+=';';
59      print(outString);
60      printString+=(outString);
61      printString+='\n';
62      #hinzufuegen einer transition
63    elif 'loop' in node.tag:
64      print(indentString+node.tag);
65      loopstack.append('p'+str(counter));
66      #hinzufuegen eines pfades zwischen dieser transition und dieser
            transition
67    elif 'choose' in node.tag:
68      print(indentString+node.tag);
69      i=0;
70      while i<len(list(node)):
71        startstack.append('p'+str(counter));
72        i+=1;
73      #counter+=1;
74      #hinzufuegen eines places
75    elif 'alternative' in node.tag:
76      print(indentString+node.tag);
77      print('trans '+'x_alternative');
78      printString+=('trans '+'x_alternative');
79      printString+='\n';
80      x=startstack.pop();
81      print('  in '+x);
82      printString+=('  in '+x);
83      printString+='\n';
84      counter+=1;
85      print('  out p'+str(counter)+';');
86      printString+=('  out p'+str(counter)+';');
87      printString+='\n';
88      endstack.append('p'+str(counter));
89      exception=True;
90      print(endstack);
91      #hinzufuegen pfad zwischen place von choose und (neuer) transition
92    elif 'otherwise' in node.tag:
93      print(indentString+node.tag);
94      print('trans '+'x_alternative');
95      printString+=('trans '+'x_alternative');
96      printString+='\n';
97      x=startstack.pop();
98      print('  in '+x);
99      printString+=('  in '+x);
100     printString+='\n';
101     counter+=1;
102     print('  out p'+str(counter)+';');
103     printString+=('  out p'+str(counter)+';');
```

```
104     printString+='\n';
105     endstack.append('p'+str(counter));
106     exception=True;
107     print(endstack);
108     #hinzufuegen pfad zwischen place von choose und (neuer) transition
109   elif 'call' in node.tag:
110     print(indentString+node.tag);
111     print('trans_'+node[0][0].text.replace('_','').replace('?','')+'_'
            +node.attrib['id']+'_'+endpointToUrl[node.attrib['endpoint']]+
            '_start');
112     printString+=('trans_"'+node[0][0].text.replace('_','').replace('?
            ','')+'_'+node.attrib['id']+'_'+endpointToUrl[node.attrib['
            endpoint']]+'_start"');
113     printString+='\n';
114     print('__in_p'+str(counter));
115     printString+=('__in_p'+str(counter));
116     printString+='\n';
117     counter+=1;
118     print('__out_p'+str(counter)+';');
119     printString+=('__out_p'+str(counter)+';');
120     printString+='\n';
121
122     print('trans_'+node[0][0].text.replace('_','').replace('?','')+'_'
            +node.attrib['id']+'_'+endpointToUrl[node.attrib['endpoint']]+
            '_complete');
123     printString+=('trans_"'+node[0][0].text.replace('_','').replace('?
            ','')+'_'+node.attrib['id']+'_'+endpointToUrl[node.attrib['
            endpoint']]+'_complete"');
124     printString+='\n';
125     print('__in_p'+str(counter));
126     printString+=('__in_p'+str(counter));
127     printString+='\n';
128     counter+=1;
129     print('__out_p'+str(counter)+';');
130     printString+=('__out_p'+str(counter)+';');
131     printString+='\n';
132
133     if not firstOfBranch:
134       endstack.pop();
135     endstack.append('p'+str(counter));
136     print(endstack);
137     #counter+=1;
138     #hinzufuegen von transition mit id
139   elif 'manipulate' in node.tag:
140     print(indentString+node.tag);
141     print('trans_'+node.attrib['label'].replace('_','').replace('?','
            )+'_'+node.attrib['id']);
142     printString+=('trans_'+node.attrib['label'].replace('_','').
            replace('?','')+'_'+node.attrib['id']);
143     printString+='\n';
144     print('__in_p'+str(counter));
145     printString+=('__in_p'+str(counter));
146     printString+='\n';
147     counter+=1;
148     print('__out_p'+str(counter)+';');
149     printString+=('__out_p'+str(counter)+';');
150     printString+='\n';
151     if not firstOfBranch:
152       endstack.pop();
153     endstack.append('p'+str(counter));
154     print(endstack);
155     #counter+=1;
156     #hinzufuegen von transition mit id
157   elif 'terminate' in node.tag:
158     print(indentString+node.tag);
159     print('trans_'+'x_termination');
160     printString+=('trans_'+'x_termination');
161     printString+='\n';
```

```
162     print('  in p'+str(counter));
163     printString+=('  in p'+str(counter));
164     printString+='\n';
165     counter+=1;
166     print('  out p'+str(counter)+';');
167     printString+=('  out p'+str(counter)+';');
168     printString+='\n';
169     if not firstOfBranch:
170       endstack.pop();
171     endstack.append('p'+str(counter));
172     termination=True;
173
174
175
176   if len(list(node))>0:
177     x=1;
178     for child in node:
179       if (x == 1) and (not exception):
180         processChild(child, indent+2, True);
181       else:
182         processChild(child, indent+2, False);
183       x+=1;
184   else:
185     #print('no childs');
186     pass;
187
188   if 'parallel_branch' in node.tag:
189     pass;
190   elif 'parallel' in node.tag:
191     counter+=1;
192     print('trans '+'x_closing_parallel');
193     printString+=('trans '+'x_closing_parallel');
194     printString+='\n';
195     i=0;
196     inString='  in';
197     while i<len(list(node)):
198       inString+=' '+endstack.pop();
199       if (i+1)<len(list(node)):
200         inString+=',';
201       i+=1;
202     outString+=';';
203     print(inString);
204     printString+=(inString);
205     printString+='\n';
206     print('  out p'+str(counter)+';');
207     printString+=('  out p'+str(counter)+';');
208     printString+='\n';
209     endstack.append('p'+str(counter));
210     #close parallel
211   elif 'choose' in node.tag:
212     counter+=1;
213     i=0;
214     if termination:
215       i+=1;
216       termination=False;
217     while i<len(list(node)):
218       print('trans '+'x_closing_decision');
219       printString+=('trans '+'x_closing_decision');
220       printString+='\n';
221       x=endstack.pop();
222       print('  in '+x);
223       printString+=('  in '+x);
224       printString+='\n';
225       print('  out p'+str(counter)+';');
226       printString+=('  out p'+str(counter)+';');
227       printString+='\n';
228       i+=1;
229     endstack.append('p'+str(counter));
```

```
230     #close decision
231   elif 'loop' in node.tag:
232     print('trans '+'x_closing_loop');
233     printString+=('trans '+'x_closing_loop');
234     printString+='\n';
235     print('  in p'+str(counter));
236     printString+=('  in p'+str(counter));
237     printString+='\n';
238     x=loopstack.pop();
239     print('  out '+x+';');
240     printString+=('  out '+x+';');
241     printString+='\n';
242     #close loop
243
244
245 print('hello world');
246
247 endpointToUrl={};
248 tree = ET.parse(sys.argv[1]);
249 root = tree.getroot();
250
251 for topchild in root:
252     x=1;
253     if(topchild.tag == 'description'):
254       for child in topchild[0]:
255         if x == 1:
256           processChild(child, 0, True);
257         else:
258           processChild(child, 0, False);
259         x+=1;
260     elif(topchild.tag == 'endpoints'):
261       for child in topchild:
262           regex=re.match('(\{.*\})(.*)', child.tag);
263           print(regex.group(2)+'  '+(child.tag)+'  '+str(child.text));
264           endpointToUrl.update({regex.group(2):child.text});
265
266 print(startstack);
267 print(endstack);
268 print(loopstack);
269 #print(printString);
270
271 completePrintString='';
272 i=0;
273 while i<counter:
274   completePrintString+='place p'+str(i);
275   if i==0:
276     completePrintString+=' init 1'
277   completePrintString+=';\n';
278   i+=1;
279
280 completePrintString+=printString;
281 print(completePrintString);
282
283 with open(sys.argv[2], 'w') as file:
284   print(completePrintString, file=file);
285
286 print(endpointToUrl);
```

In order to create the XES log file, all log files given in YAML format as described in section 1 have to be transformed. To achieve this, the information from the YAML "event" elements containing a "cpee:lifecycle:transition" element with the value "activity/calling" or "activity/done" is written into the XES file. Depending on the "cpee:lifecycle:transition" the events are considered to be either start or end of the task. The final XES file contains all events (starting as well as ending) of all logs.

The XES file is structured into a header section, which includes extensions, global keys for events and traces and classifiers, which are used to classify the events. An additional classifier is added for the second scenario, which is the endpoint in combination with the CPEE lifecycle transition. The reason for this is explained in section 2.4. Following, are multiple traces, one for each YAML log and each trace contains multiple events, which are, as just mentioned, mapped to the starting and completing of tasks in the YAML log file. Each trace has an ID or "concept:name", a name or "cpee:name" and the UUID. Each event has a name, an endpoint (although not all events actually have an endpoint, namely all "script" tasks), an ID (which is the ID of the task in the template), the lifecycle transition ("start" or "complete"), the CPEE lifecycle transition ("activity/calling" and "activity/done") and the timestamp.

The program transforming the YAML logs to a XES file as described above is written in python (see also [27] for scenario 2 and [26] for scenario 1) and can be executed by using the following command: "python xes_map.py" - in order to execute successfully the slightly edited library XES 1.3 for python which can be installed using "pip install xes" (the edited version "xes.py" is available at [5]) as well as the file "filepaths.txt"given at [18] for scenario 1 and [19] for scenario 2 which point to the locations of the logs that should be included have to be present:

```
import yaml;
import xes;
import os;

id_uuid = {}


with open("filepaths.txt") as f:
  filepaths = f.readlines()
filepaths = [x.strip() for x in filepaths]
n=1
filedata = {filepath: open(filepath, 'r') for filepath in filepaths}
log = xes.Log()
log_set = False
for file in filedata.values():
  trees = yaml.load_all(file);
  t = xes.Trace()
  for tree in trees:
    if 'log' in tree:
      if log_set == False:
        log.extensions = [
          xes.Extension(name="Time",prefix="time",uri=tree['log']['extension']['time']),
          xes.Extension(name="Concept",prefix="concept",uri=tree['log']['extension']['concept']),
          xes.Extension(name="Organizational",prefix="org",uri=tree['log']['extension']['organisational']),
          xes.Extension(name="Lifecycle",prefix="lifecycle",uri=tree['log']['extension']['lifecycle'])
        ]
        log.global_trace_attributes = [
          xes.Attribute(type="string",key='concept:name',value=tree['log']['global']['trace']['concept:name']),
          xes.Attribute(type="string",key='cpee:name',value=tree['log']['global']['trace']['cpee:name'])
        ]
        log.global_event_attributes = [
```

```
32            xes.Attribute(type="string",key='concept:name',value=tree['
                  log']['global']['event']['concept:name']),
33            xes.Attribute(type="string",key='cpee:endpoint',value=tree['
                  log']['global']['event']['concept:endpoint']),
34            xes.Attribute(type="string",key='id:id',value=tree['log']['
                  global']['event']['id:id']),
35            xes.Attribute(type="string",key='lifecycle:transition',value
                  =tree['log']['global']['event']['lifecycle:transition'])
                  ,
36            xes.Attribute(type="string",key='cpee:lifecycle:transition',
                  value=tree['log']['global']['event']['cpee:lifecycle:
                  transition']),
37            xes.Attribute(type="date",key='time:timestamp',value="
                  1990-02-17T09:45:00.000+01:00")
38          ]
39          log.classifiers = [
40            xes.Classifier(name="Event ID Transition Classifier",keys="
                  id:id lifecycle:transition"),
41            xes.Classifier(name="MXML Legacy Classifier",keys="concept:
                  name lifecycle:transition"),
42            xes.Classifier(name="Event Name",keys="concept:name"),
43            xes.Classifier(name="Event ID",keys="id:id"),
44            xes.Classifier(name="CPEE Classifier",keys="concept:name
                  cpee:lifecycle:transition"),
45            xes.Classifier(name="CPEE Endpoint",keys="cpee:endpoint cpee
                  :lifecycle:transition")
46          ]
47          log_set= True
48        t.attributes = [
49          xes.Attribute(type="string", key="concept:name", value=tree['
                log']['trace']['concept:name']),
50          xes.Attribute(type="string", key="cpee:name", value=tree['log'
                ]['trace']['cpee:name']),
51          xes.Attribute(type="string", key="cpee:uuid", value=tree['log'
                ]['trace']['cpee:uuid'])
52        ]
53      if 'event' in tree:
54        if tree['event']['cpee:lifecycle:transition'] == 'activity/
              calling' or tree['event']['cpee:lifecycle:transition'] == '
              activity/done':
55          e = xes.Event()
56          endpoint=""
57          if('concept:endpoint' in tree['event']):
58            endpoint = tree['event']['concept:endpoint']
59          e.attributes = [
60            xes.Attribute(type="string", key="concept:name", value=tree[
                  'event']['concept:name']),
61            xes.Attribute(type="string", key="cpee:endpoint", value=
                  endpoint),
62            xes.Attribute(type="string", key="id:id", value=tree['event'
                  ]['id:id']),
63            xes.Attribute(type="string", key="lifecycle:transition",
                  value=tree['event']['lifecycle:transition']),
64            xes.Attribute(type="string", key="cpee:lifecycle:transition"
                  , value=tree['event']['cpee:lifecycle:transition']),
65            xes.Attribute(type="date", key="time:timestamp", value=tree[
                  'event']['time:timestamp'])
66          ]
67          t.add_event(e)
68    log.add_trace(t)
69    print(str(n) + " of " + str(len(filepaths)) + " logs parsed.")
70    n=n+1
71
72
73 open("logs.xes", "w").write(str(log))
```

2.2 Fitness

Fitness determines how well a process log is described by the given process template. The fitness value is between 0 and 1. A value of 1 shows that a log is completely conformant to the template while lower values indicate that the execution of the log is not covered well by the template. Different kinds of fitness values exist. The ones used for this work are:

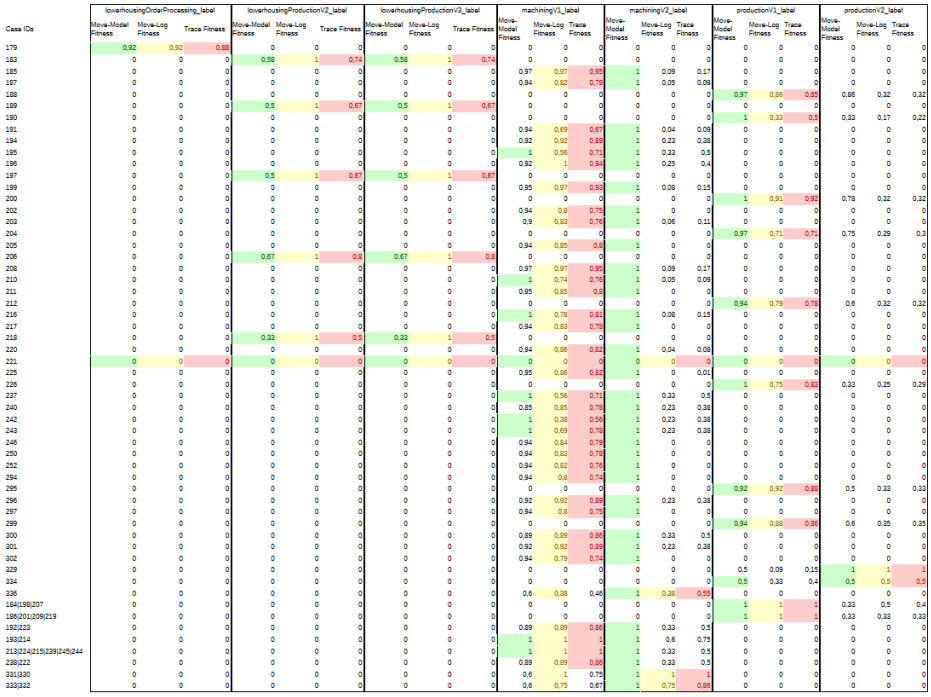
- **Move-Model Fitness** is the value of correct model moves divided by the overall number of model moves during the replay.
- **Move-Log Fitness** is the value of correct log moves divided by the overall number of log moves during the replay.
- **Trace Fitness** is the cost-based fitness value as described in [1, p.7]. This is a value obtained by subtracting the raw fitness cost divided by the maximum fitness cost from 1.

Executing the conformance check as provided in the "Replay a Log on Petri Net for Conformance Analysis" from the "PNetReplayer" ProM package on the petri net created following the rules given in section 2.1 (and creating final markings as well as making helper transitions invisible) and the XES log with standard parameters (which means choosing the option "penalize improper completion", choosing "A* Cost-based Fitness Express with ILP" as the algorithm, and setting all "Move on Model Costs" and "Move on Log Costs" to 1) makes it possible to obtain a CSV file with the abovementioned fitness values. These are used to create the tables given in section 2.3 and 2.4 showing the results. They contain one line for each log trace (or group of log traces if there are identical ones). The columns represent one of the three fitness values given above (for each of the templates). The maximum value for each line is highlighted. (green for move-model fitness, yellow for move-log fitness and red for trace fitness).

2.3 Application to Log Data - Scenario 1

For the first scenario, the events from the XES log are mapped to the petri net transitions using the labels of the tasks (e.g. Detect Lowerhousing Production Start) combined with the information if it is starting or ending.

The result table given in Fig. 2 shows for each log how well it fits each process template and highlights the maximum value of each characteristic by giving the corresponding cell a green (maximum move-model fitness), yellow (maximum move-log fitness), or red (maximum trace fitness) background color. Based on this, one can decide an instance of which template the log is - sometimes the template of the maximum value of move-model fitness differs from the template of the maximum value of the other two characteristics, this seems to be based on a combination of short templates (e.g. MachiningV2) combined with longer logs (e.g. log of MachiningV1). If this happens, the model is moved in a correct way going through the short template while skipping many log events (which is not important for the move-model fitness) - for the longer MachiningV1 template which the trace is actually based on there is a small number of steps where a



Case IDs	lowerhousingOrderProcessing_label			lowerhousingProductionV2_label			lowerhousingProductionV3_label			machiningV1_label			machiningV2_label			productionV1_label			productionV2_label		
	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness
179	0,92	0,92	0,89	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
183	0	0	0	0,58	1	0,74	0,58	1	0,74	0	0	0	0	0	0	0	0	0	0	0	0
185	0	0	0	0	0	0	0	0	0	0,97	0,97	0,95	1	0,09	0,17	0	0	0	0	0	0
187	0	0	0	0	0	0	0	0	0	0,94	0,82	0,79	1	0,05	0,09	0	0	0	0	0	0
188	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,97	0,86	0,83	0,86	0,32	0,32
189	0	0	0	0,5	1	0,67	0,5	1	0,67	0	0	0	0	0	0	0	0	0	0	0	0
190	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0,33	0,5	0,33	0,17	0,22
191	0	0	0	0	0	0	0	0	0	0,94	0,69	0,67	1	0,04	0,08	0	0	0	0	0	0
194	0	0	0	0	0	0	0	0	0	0,92	0,92	0,89	1	0,23	0,38	0	0	0	0	0	0
195	0	0	0	0	0	0	0	0	0	1	0,56	0,71	1	0,33	0,5	0	0	0	0	0	0
196	0	0	0	0	0	0	0	0	0	0,92	1	0,94	1	0,25	0,4	0	0	0	0	0	0
197	0	0	0	0,5	1	0,67	0,5	1	0,67	0	0	0	0	0	0	0	0	0	0	0	0
199	0	0	0	0	0	0	0	0	0	0,95	0,97	0,93	1	0,08	0,15	0	0	0	0	0	0
200	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0,91	0,82	0,78	0,32	0,32
202	0	0	0	0	0	0	0	0	0	0,94	0,8	0,75	1	0	0	0	0	0	0	0	0
203	0	0	0	0	0	0	0	0	0	0,9	0,83	0,76	1	0,06	0,11	0	0	0	0	0	0
204	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,97	0,71	0,71	0,75	0,29	0,3
205	0	0	0	0	0	0	0	0	0	0,94	0,85	0,8	1	0	0	0	0	0	0	0	0
206	0	0	0	0,67	1	0,8	0,67	1	0,8	0	0	0	0	0	0	0	0	0	0	0	0
208	0	0	0	0	0	0	0	0	0	0,97	0,97	0,95	1	0,09	0,17	0	0	0	0	0	0
210	0	0	0	0	0	0	0	0	0	1	0,74	0,76	1	0,05	0,09	0	0	0	0	0	0
211	0	0	0	0	0	0	0	0	0	0,95	0,85	0,8	1	0	0	0	0	0	0	0	0
212	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,94	0,79	0,78	0,6	0,32	0,32
216	0	0	0	0	0	0	0	0	0	1	0,78	0,81	1	0,08	0,15	0	0	0	0	0	0
217	0	0	0	0	0	0	0	0	0	0,94	0,83	0,78	1	0	0	0	0	0	0	0	0
218	0	0	0	0,33	1	0,5	0,33	1	0,5	0	0	0	0	0	0	0	0	0	0	0	0
220	0	0	0	0	0	0	0	0	0	0,94	0,86	0,82	1	0,04	0,08	0	0	0	0	0	0
221	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
225	0	0	0	0	0	0	0	0	0	0,95	0,86	0,82	1	0	0,01	0	0	0	0	0	0
226	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0,75	0,82	0,33	0,25	0,29
227	0	0	0	0	0	0	0	0	0	1	0,56	0,71	1	0,33	0,5	0	0	0	0	0	0
240	0	0	0	0	0	0	0	0	0	0,85	0,85	0,76	1	0,23	0,38	0	0	0	0	0	0
242	0	0	0	0	0	0	0	0	0	1	0,38	0,56	1	0,23	0,38	0	0	0	0	0	0
243	0	0	0	0	0	0	0	0	0	1	0,69	0,78	1	0,23	0,38	0	0	0	0	0	0
246	0	0	0	0	0	0	0	0	0	0,94	0,84	0,79	1	0	0	0	0	0	0	0	0
250	0	0	0	0	0	0	0	0	0	0,94	0,83	0,78	1	0	0	0	0	0	0	0	0
252	0	0	0	0	0	0	0	0	0	0,94	0,82	0,78	1	0	0	0	0	0	0	0	0
254	0	0	0	0	0	0	0	0	0	0,94	0,8	0,74	1	0	0	0	0	0	0	0	0
295	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,92	0,92	0,88	0,5	0,33	0,33
296	0	0	0	0	0	0	0	0	0	0,92	0,92	0,89	1	0,23	0,38	0	0	0	0	0	0
297	0	0	0	0	0	0	0	0	0	0,94	0,8	0,75	1	0	0	0	0	0	0	0	0
299	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,94	0,88	0,86	0,6	0,35	0,35
300	0	0	0	0	0	0	0	0	0	0,89	0,89	0,86	1	0,33	0,5	0	0	0	0	0	0
301	0	0	0	0	0	0	0	0	0	0,92	0,92	0,89	1	0,23	0,38	0	0	0	0	0	0
302	0	0	0	0	0	0	0	0	0	0,94	0,79	0,74	1	0	0	0	0	0	0	0	0
329	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,5	0,09	0,15	1	1	1
334	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,5	0,33	0,4	0,5	0,5	0,5
336	0	0	0	0	0	0	0	0	0	0,6	0,38	0,46	1	0,38	0,55	0	0	0	0	0	0
184|198|207	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0,33	0,5	0,4
186|201|209|219	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0,33	0,33	0,33
192|223	0	0	0	0	0	0	0	0	0	0,89	0,89	0,86	1	0,33	0,5	0	0	0	0	0	0
193|214	0	0	0	0	0	0	0	0	0	1	1	1	1	0,6	0,75	0	0	0	0	0	0
213|224|215|239|245|244	0	0	0	0	0	0	0	0	0	1	1	1	1	0,33	0,5	0	0	0	0	0	0
238|222	0	0	0	0	0	0	0	0	0	0,89	0,89	0,86	1	0,33	0,5	0	0	0	0	0	0
331|330	0	0	0	0	0	0	0	0	0	0,6	1	0,75	1	1	1	0	0	0	0	0	0
333|332	0	0	0	0	0	0	0	0	0	0,6	0,75	0,67	1	0,75	0,88	0	0	0	0	0	0

Fig. 2: Conformance Checking Results Using Labels For Event/Transition Mapping

task in the model is skipped because some events might be in the wrong order or missing which leads to a slightly lower move-model fitness. These effects do not occur in the move-log fitness and trace fitness so it is concluded that these two features should be the ones to base the decision on (conveniently for all traces these two do not contradict each other when deciding which template a trace is based on so it is not necessary to think further about which of them is more significant). The results of this analysis match the ones of the manual analysis (given in templates.xlsx - available at [25]). Therefore it is concluded that the goal of identifying the underlying template of a log using conformance checking is working as expected.

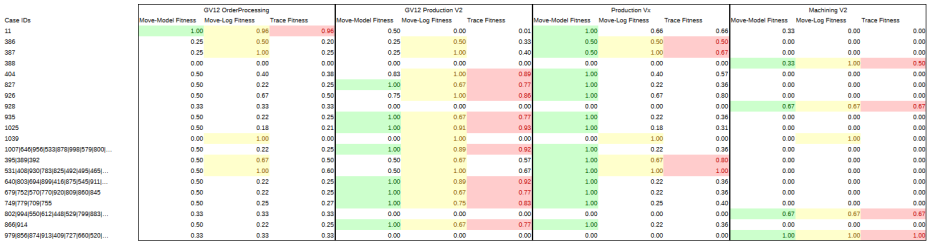
2.4 Application to Log Data - Scenario 2

In addition to the approach used in the first scenario described in section 2.3, not only the labels were used for mapping events in the log to tasks in the petri net. The two characteristics which were used to perform the abovementioned mapping for the second scenario are:

- using labels (e.g. “Detect GV12 Production Start”) as for the first scenario
- using endpoints (e.g. “https://centurio.work/flow/start/url/”)

Case IDs	GV12 OrderProcessing			GV12 Production V2			Production Vx			Machining V2		
	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness
11	1.00	0.94	0.94	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
386	0.00	0.00	0.00	0.25	0.50	0.33	0.00	0.00	0.00	0.00	0.00	0.00
387	0.00	0.00	0.00	0.00	0.00	0.00	0.50	1.00	0.67	0.00	0.00	0.00
388	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.33	1.00	0.50
404	0.00	0.00	0.00	0.67	0.80	0.67	0.00	0.00	0.00	0.00	0.00	0.00
827	0.00	0.00	0.00	1.00	0.67	0.77	0.00	0.00	0.00	0.00	0.00	0.00
926	0.00	0.00	0.00	0.50	0.67	0.57	0.00	0.00	0.00	0.00	0.00	0.00
928	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.67	0.67	0.67
935	0.00	0.00	0.00	1.00	0.67	0.77	0.00	0.00	0.00	0.00	0.00	0.00
1025	0.00	0.00	0.00	1.00	0.91	0.83	0.00	0.00	0.00	0.00	0.00	0.00
1039	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00
1007|646|956|533|878|998|579|800|...	0.00	0.00	0.00	1.00	0.89	0.92	0.00	0.00	0.00	0.00	0.00	0.00
395|389|392	0.00	0.00	0.00	0.50	0.67	0.57	0.00	0.00	0.00	0.00	0.00	0.00
531|408|930|783|825|492|495|465|...	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00
640|803|694|899|416|875|545|911|...	0.00	0.00	0.00	1.00	0.89	0.92	0.00	0.00	0.00	0.00	0.00	0.00
679|752|570|770|920|809|860|845	0.00	0.00	0.00	1.00	0.67	0.77	0.00	0.00	0.00	0.00	0.00	0.00
749|779|709|755	0.00	0.00	0.00	1.00	0.75	0.83	0.00	0.00	0.00	0.00	0.00	0.00
802|994|550|612|448|529|799|883|...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.67	0.67	0.67
866|914	0.00	0.00	0.00	1.00	0.67	0.77	0.00	0.00	0.00	0.00	0.00	0.00
979|856|874|913|409|727|660|520|...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00

Fig. 3: Conformance Checking Results Using Labels For Event/Transition Mapping



Case IDs	GV12 OrderProcessing			GV12 Production V2			Production Vx			Machining V2		
	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness	Move-Model Fitness	Move-Log Fitness	Trace Fitness
11	1.00	0.96	0.96	0.50	0.00	0.01	1.00	0.66	0.66	0.33	0.00	0.00
386	0.25	0.50	0.20	0.25	0.50	0.33	0.50	0.50	0.50	0.00	0.00	0.00
387	0.25	1.00	0.25	0.25	1.00	0.40	0.50	1.00	0.67	0.00	0.00	0.00
388	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.33	1.00	0.50
404	0.50	0.40	0.38	0.83	1.00	0.89	1.00	0.40	0.57	0.00	0.00	0.00
827	0.50	0.22	0.25	1.00	0.67	0.77	1.00	0.22	0.36	0.00	0.00	0.00
926	0.50	0.67	0.50	0.75	1.00	0.86	1.00	0.67	0.80	0.00	0.00	0.00
928	0.33	0.33	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.67	0.67	0.67
935	0.50	0.22	0.25	1.00	0.67	0.77	1.00	0.22	0.36	0.00	0.00	0.00
1025	0.50	0.18	0.21	1.00	0.91	0.93	1.00	0.18	0.31	0.00	0.00	0.00
1039	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00
1007|646|956|533|878|998|579|800|...	0.50	0.22	0.25	1.00	0.89	0.92	1.00	0.22	0.36	0.00	0.00	0.00
395|389|392	0.50	0.67	0.50	0.50	0.67	0.57	1.00	0.67	0.80	0.00	0.00	0.00
531|408|930|783|825|492|495|465|...	0.50	1.00	0.60	0.50	1.00	0.67	1.00	1.00	1.00	0.00	0.00	0.00
640|803|694|899|416|875|545|911|...	0.50	0.22	0.25	1.00	0.89	0.92	1.00	0.22	0.36	0.00	0.00	0.00
679|752|570|770|920|809|860|845	0.50	0.22	0.25	1.00	0.67	0.77	1.00	0.22	0.36	0.00	0.00	0.00
749|779|709|755	0.50	0.25	0.27	1.00	0.75	0.83	1.00	0.25	0.40	0.00	0.00	0.00
802|994|550|612|448|529|799|883|...	0.33	0.33	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.67	0.67	0.67
866|914	0.50	0.22	0.25	1.00	0.67	0.77	1.00	0.22	0.36	0.00	0.00	0.00
979|856|874|913|409|727|660|520|...	0.33	0.33	0.33	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00

Fig. 4: Conformance Checking Results Using Endpoints For Event/Transition Mapping

Using labels leads to a good solution in the first scenario. Using endpoints is an alternative characteristic that can be used. Therefore, the results of both characteristics are compared to each other to show the advantages and disadvantages.

As can be seen in the results given in Fig. 3 and 4, the matching of the logs to the corresponding templates is similar for using labels and endpoints as characteristic for the mapping of events to petri net transitions. The following advantages are present when using labels

- all logs (apart from the plain instance - log number 1039) are correctly assigned to the corresponding templates
- the results obtained are clear (i.e. the fitness values of a log for all but the corresponding template is 0)

and endpoints respectively

- the results are more stable (i.e. the result is not dependent on the label of the tasks)
- endpoints describe actual functionality of a task

Clearly, there are also some disadvantages for both methods. For using labels these are

- unstable if labels are changed (e.g. “QC Shop Floor” is renamed to “Manually Measure” at some point)

– labels are not linked to the functionality of a task

and for endpoints:

– if endpoints are used, it is oftentimes difficult to determine the process template (e.g. "start/url" as generic template is a problem)
– the results are less clear (higher fitness for false templates) and sometimes even the false template is chosen

2.5 Lessons Learned

Overall, as the endpoints provide a much more stable result which is also based on the functionality of tasks, this would be the desired characteristic. But there is also a major drawback using it which comes from the endpoints being generic and therefore being used in many different process models (see for example the "start/url" endpoint). In order to tackle this problem, one could think of including the parameters of these endpoints to get a more detailed idea of the functionality performed. Unluckily, this is more difficult as it seems at first because there is no distinction between parameters for a call to an endpoint that are fixed and ones that can (at least theoretically) be changed at runtime (i.e. by using a data element which is obtained during the process execution) and therefore the usage of the information given in the template models may not always be the one which is also used at execution time. Consequently, using the value of data elements (and therefore also parameters of endpoints) is not possible because they may not be defined in the template at all but at a later point during the execution.

3 Classification and Clustering of Log Data

3.1 Extracting Machining Data From YAML Logs

Since the log data is available in the YAML format, several steps have to be taken to transform them into data which can be used for classification and clustering. In order to achieve this, the logs are first transformed into CSV files taking only the "Fetch" tasks with the "activity/receiving" lifecycle transition into consideration as these are the only ones containing machining data. The data extracted from the logs contains the following information that is used in the CSV files: "source", "name", "description", "path", "value", "timestamp", "StatusCode", "ServerTimestamp", "VariantType", "ClientHandle". Afterwards, it has to be defined which logs failed and which ones were successful. This differs for the two scenarios. In the first one the goal is to find out if the machining log is the last one spawned by the parent "Production" process which means that it is the one finishing successfully. For the second scenario successful logs are the ones which pass the measurements (the code for extracting the measurements from the YAML logs is available at [22]) - different measurements are used for classification as described in section 3.8. Using the information from the CSV

files and the definition of successful logs, the data is imported into R. In order to represent all logs in the same way, some common parameters need to be chosen. This is done by only using logs which are long enough, finding parameters which occur in all of them and then choosing the ones which occur often enough for meaningful analysis. More details about the selection of logs and parameters are given in the sections where the application of the methods to the data is described (3.7 and 3.8).

The python code for extracting the machining data from the YAML log files and write them into a CSV file is given below. It is also provided at [14] for the second scenario (at [13] for the first scenario) and executed using the command "python csv_map.py" which needs the file "machiningFilepaths.txt" (available at [21]) or for the first scenario "filepaths_machining.txt" (available at [20]) to identify the machining logs containing the data.

```
import yaml;
import os;
import csv;
import re;
import sys;

csvData = [["Id","source","name","description","path","value","
    timestamp","StatusCode","ServerTimestamp","VariantType","
    ClientHandle"]]

with open("machiningFilepaths.txt") as f:
  filepaths = f.readlines()
filepaths = [x.strip() for x in filepaths]
n=1
#filepaths = ['logs/production/acc7d2e4-f949-4e9b-a99a-afe3469cbbe9.
    xes.yaml']

filedata = {filepath: open(filepath, 'r') for filepath in filepaths}

#print(filedata)

for file in filedata.values():
  print(file.name)
  logname=str(n)
  trees = yaml.load_all(file);
  for tree in trees:
    if 'log' in tree:
        logname=tree['log']['trace']['concept:name']
    if 'event' in tree:
      if tree['event']['cpee:lifecycle:transition'] == 'activity/
          receiving' and tree['event']['concept:name'] == 'Fetch':
        if 'list' in tree['event']:
          for data_r in tree['event']['list']['data_receiver']:
            for data in data_r['data']:
              id=""
              source=""
              name=""
              description=""
              path=""
              value=""
              timestamp=""
              statusCode=""
              serverTimestamp=""
              variantType=""
              clientHandle=""
              if 'ID' in data:
                id=data['ID']
              if 'source' in data:
```

```
45                  source=data['source']
46                if 'name' in data:
47                  name=data['name']
48                if 'description' in data:
49                  description=data['description']
50                if 'path' in data:
51                  path=data['path']
52                if 'value' in data:
53                  value=str(data['value']).rstrip()
54                if 'timestamp' in data:
55                  timestamp=data['timestamp']
56                if 'meta' in data:
57                  if 'StatusCode' in data['meta']:
58                    statusCode=data['meta']['StatusCode']
59                  if 'ServerTimestamp' in data['meta']:
60                    serverTimestamp=data['meta']['ServerTimestamp']
61                  if 'VariantType' in data['meta']:
62                    variantType=data['meta']['VariantType']
63                  if 'ClientHandle' in data['meta']:
64                    clientHandle=data['meta']['ClientHandle']
65                csvNewData = [id,source,name,description,path,value,
                      timestamp,statusCode,serverTimestamp,variantType,
                      clientHandle]
66                csvData.append(csvNewData)
67            else:
68              csvNewData = ["","","","","","",tree['event']['time:
                    timestamp'],"","","","",""]
69              csvData.append(csvNewData)
70    print(str(n) + "␣of␣" + str(len(filepaths)) + "␣logs␣parsed.")
71    csvString=""
72    for v in csvData:
73      line='*'.join(str(r) for r in v)
74      csvString+=line+"\n"
75    open("log"+str(logname)+".csv", "w").write(csvString)
76    csvData=[["Id","source","name","description","path","value","
          timestamp","StatusCode","ServerTimestamp","VariantType","
          ClientHandle"]];
77    n=n+1
78
79 sys.exit(0);
```

3.2 Hierarchical Clustering

Hierarchical clustering is a method of building clusters by either starting with each observation in its own cluster or with all observations in one cluster and then merge the ones being closest to each other using some distance metric (or split into a number of clusters based on the distance in case one big cluster is used as starting point). In order to cluster the data, only the features derived from the logs as described earlier in section 3.1 were used without looking at if the log is successful or not. In order to find the appropriate number of clusters, a scree plot is created and clustering is then performed for promising cluster numbers. Additionally, a silhouette plot is created so that statements about the quality of clustering can be made. This method is only performed for the first scenario.

3.3 K-Means Clustering

K-Means clustering is a method where the number of clusters is defined before the algorithm is performed. The cluster centers are then randomly assigned and

shift iteratively based on the mean of the contained points. It is performed with the same data set as in section 3.2 using the same number of clusters as for the hierarchical clustering to check if the usage of different clustering methods has any effect on the result. A silhouette plot is also created for each clustering with different numbers of clusters. As for hierarchical clustering, this method is only performed for the first scenario.

3.4 Feature Selection with Random Forest

Classifying data sets with a lot of feature variables in relation to the amount of data points, comes with the need to reduce this amount of feature variables. This reduces the amount of overfitting and makes the training of data and interpretation of results easier ([3]) which is relevant for the data sets of both scenarios, especially the first one. Goal of the feature selection is to find out which feature variables are the most important, meaning which contribute the most to the corresponding class of the data.

For the feature selection an automatic method provided by the "caret" package in R is used, namely recursive feature elimination. This algorithm builds many models with the given classification algorithm and different subsets of feature variables and compares the accuracy ([6]). In this case the random forest algorithm is used to compare the models with each other. The random forest algorithm works by creating many randomly generated decision trees. Each decision tree of this forest is used for classifying the given data. The class which gets the majority of predictions of the trees is used as the final classification outcome of the forest. The advantage of the random forest algorithm is that it trains and evaluates very fast, especially with big amounts of data and is performing well in recognizing important feature variables ([4]).

3.5 Support Vector Machines

Support vector machines is a method to classify data. This is done by constructing one or multiple hyperplanes which separate data points into different classes. This is the primary method of classification for the data sets given. The usage of SVM was done in R, using linear, radial, sigmoid and polynomial kernels. In order to perform classification the function "svm" from the R package "e1071" is used (for more information on this package see [28]).

3.6 Naive Bayes

Naive Bayes is another method to classify data, based on probabilities. This was used as a comparison to SVM, to see if using a different classifier has significant effects on the classification. In order to perform classification the function "naiveBayes" from the R package "e1071" is used (for more information on this package see [28]).

3.7 Application to Log Data - Scenario 1

Every log has not only differences in the number of variables measured but also in the length of the log (i.e. the number of data points) in general. Therefore, it is first analysed how many data points are contained in each log in order to find the ones which can be used for the analysis later on.

Obviously, the logs having only few data elements are very difficult to use for the analysis task and therefore only the logs which have more or equal to 100 data elements can be used. As 6 of 47 logs do not meet this prerequisite, only 41 logs are remaining after the filtering. The next task is to look at the parameters present in all files and then the minimum number of these IDs within the files that can be used (which means files which have ≥ 100 data elements) are calculated. Afterwards, it has to be decided which data points are used (the last 10 data elements of each usable variable in this case). Therefore, a log is represented in the following way: the last 10 values of the variables following variables are used:

- ns=2;s=/Channel/MachineAxis/aaLoad[u1,1]
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,2]
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,3]
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,1]
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,2]
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,3]
- ns=2;s=/Channel/Spindle/driveLoad

Therefore, a log is represented by 70 ($7 * 10$) parameters (+1 if the success or no success flag is also taken into consideration). This data is then used to perform the data analysis.
Since the amount of feature variables in the first data set is really high with 71 in comparison to the amount of data logs with 41, the risk of overfitting the data when trying to classify them has to be considered (for example using SVM as algorithm for classifying the data, the accuracy for the test and training data is, depending on the seed, nearly always or close to 100%, which is probably an unrealistic value and that model could not be used for other data). Therefore, before classification can be applied, features must be selected. This is done using the method described in section 3.4.

In order to get the results presented below the R code given at [10] has to be executed. Fig. 5 created with this code, shows the root-mean-squared-error in comparison to the number of variables used in the model. In this case using 8 variables gives the lowest root-mean-squared-error and therefore is a good indicator, which and how many feature variables are sufficient, to create a model out of it. The variables used are (the number after the dot indicates which of the values within the last 10 - this means that ".1" is the 10th value and ".10" is the first value counting from the back):

- ns=2;s=/Channel/MachineAxis/aaLoad[u1,1].7
- ns=2;s=/Channel/Spindle/driveLoad.8
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,1].7

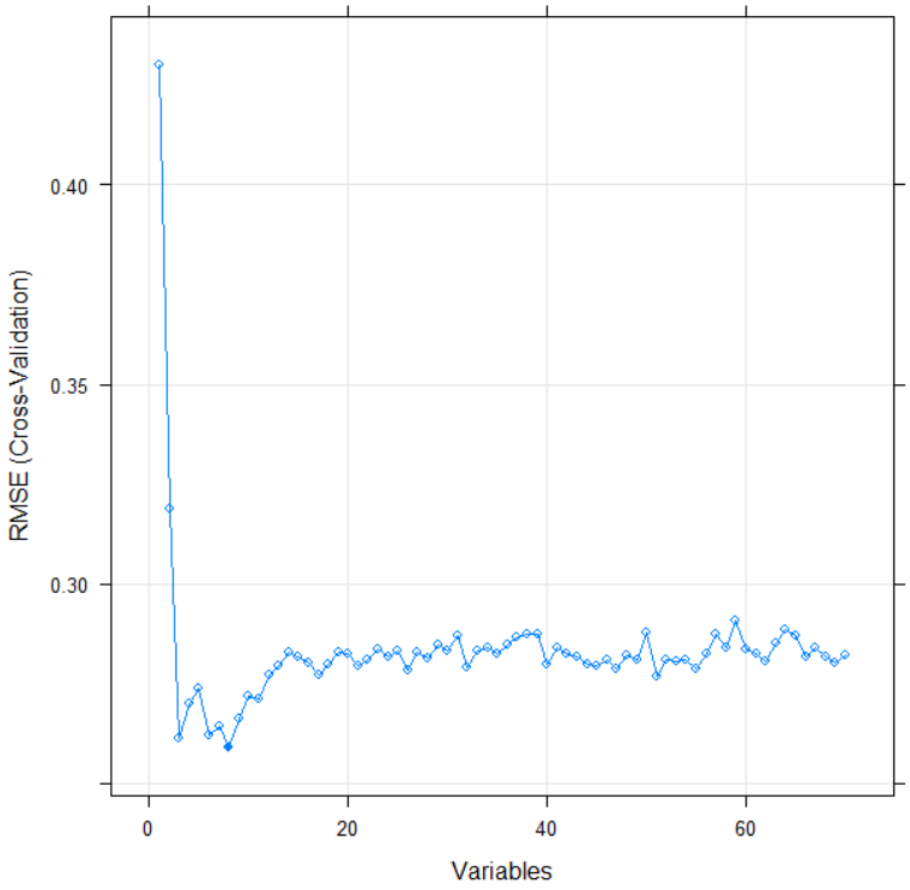


Fig. 5: Recursive Feature Elimination

- ns=2;s=/Channel/MachineAxis/aaLoad[u1,1].6
- ns=2;s=/Channel/Spindle/driveLoad.7
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,1].8
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,1].5
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,1].10

Out of those feature variables a new data frame is created. In the next step the data is split into a training set (75%, 31 logs) and a test set (25%, 10 logs).

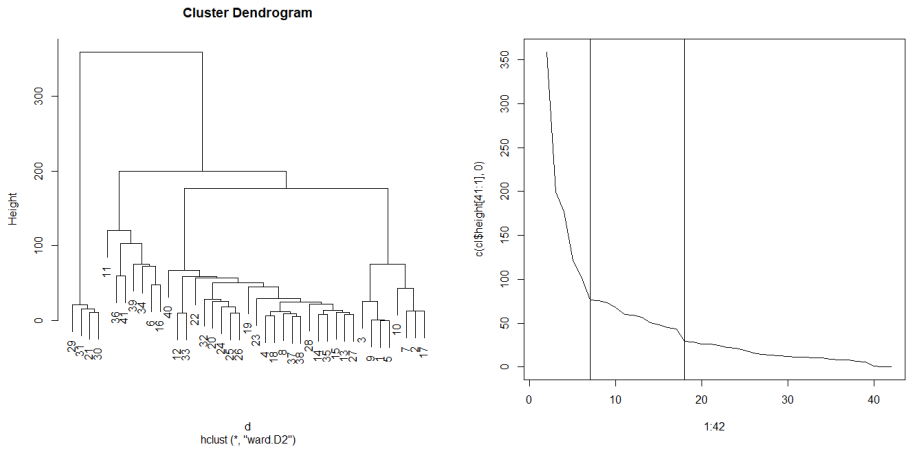
For classification different algorithms are used, to give an idea about the accuracy of classification and if the models could be viable for classifying the logs.

The accuracy given in Tab. 1 using naive bayes with 77,42% for the training set and 80% for the test set, seems worse than the SVMs. This could indicate that

	SVM				Naive Bayes
kernel	linear	radial	sigmoid	polynomial	-
cost	1	1	1	1	-
gamma	0.125	0.125	0.125	0.125	-
degree	-	-	-	3	-
coef.0	-	-	0	0	-
# support vectors	9	12	12	13	-
accuracy training set	96.77%	93.55%	93.55%	87.10%	77.42%
accuracy test set	90.00%	90.00%	90.00%	80.00%	80.00%

Tab. 1: Classification Results With Different Kernels For Scenario 1

SVM is a good algorithm for this data set. The accuracies for the training set with 93,55% for both radial and sigmoid and 87,1% for polynomial are all decently high, but still lower than SVM with a linear kernel. Furthermore, the accuracy for the test set with 90% for both radial and sigmoid and 80% for polynomial, are equal or a bit less than the SVM with linear kernel, but regarding the fact that there are only 10 data sets in the test set, jumps in accuracy are very high for single correct or incorrect predictions.

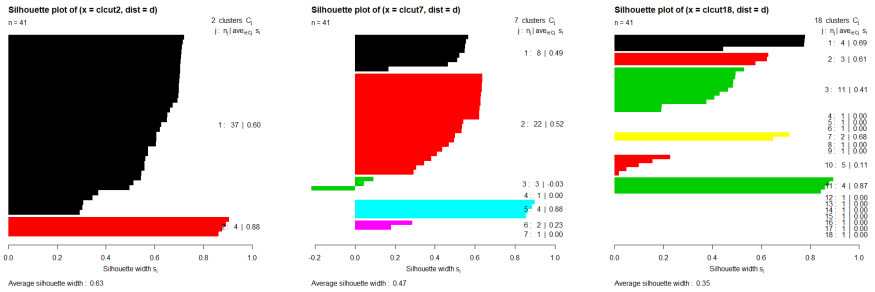


(a) Hierarchical Clustering (b) Scree Plot

Fig. 6: Hierarchical Clustering Plots

In addition to classification, clustering is performed using the code given in [11]. The scree plot (Fig. 6b) indicates that 7 or 18 clusters should be used because these are the locations where an "elbow" can be seen. Additionally, the clustering is performed for 2 clusters because there are two cases in the original problem statement (success or not success).

The silhouette plots in Fig. 7 are created for the results with different numbers of clusters.



(a) 2 Clusters (b) 7 Clusters (c) 18 Clusters

Fig. 7: Silhouette Plot For Different Numbers of Clusters Using Hierarchical Clustering

If a class is assigned to each cluster (the one which has the highest number of points inside the cluster is chosen), an estimation of the quality of the clustering is obtained:

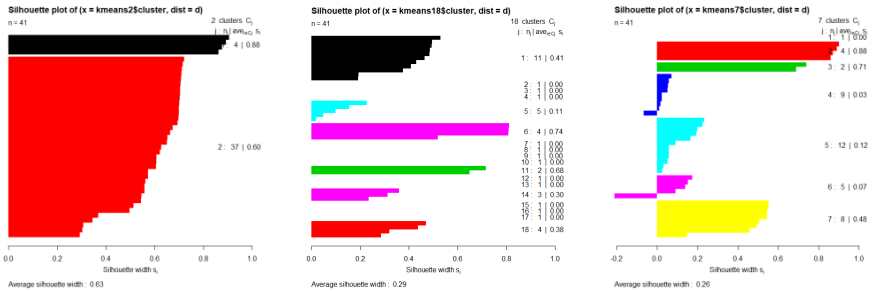
- 2 clusters: 68.83%
- 7 clusters: 92.68%
- 18 clusters: 97.56%

Obviously, the clustering using 18 clusters is the most accurate one but looking at the ratio of clusters to number of logs, and the small number of logs in most of the clusters this approach looks like overfitting. Also, the average silhouette width decreases when more clusters are used which indicates that the points are not as well located in the cluster compared to fewer clusters.

Doing clustering with the same data described above but using kmeans leads to the following results:

- 2 clusters: 68.83%
- 7 clusters: 92.68%
- 18 clusters: 95.12%

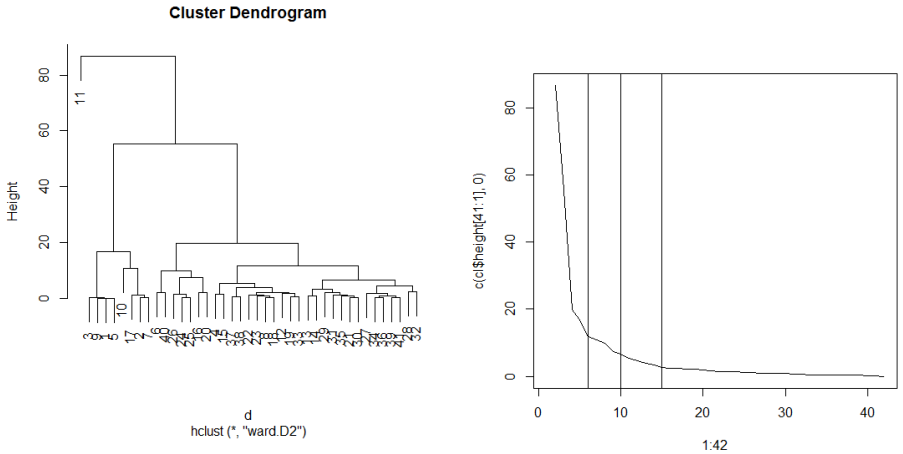
The results are very similar to the ones above concerning accuracy therefore again overfitting for 18 clusters (and maybe even 7) has to be taken into account. A notable difference to the hierarchical clustering is that the average silhouette width for 7 and 18 clusters is lower.



(a) 2 Clusters (b) 7 Clusters (c) 18 Clusters

Fig. 8: Silhouette Plot For Different Numbers of Clusters Using K-Means Clustering

After performing the clustering with the data described above, the question arised how clustering would look if less features are used – therefore clustering is done again this time not using the last 10 values of each "good" parameter but only the first one of them (i.e. the 10th value counting from the last value). The results gathered when executing the R code available at [12] are given in Fig. 9.



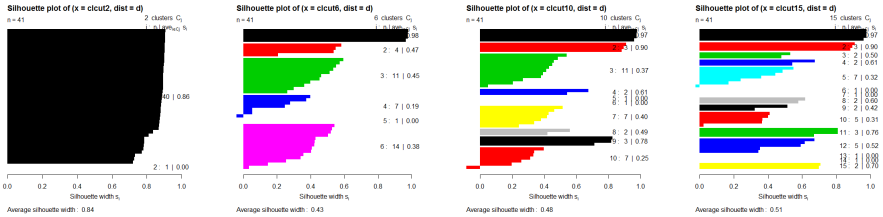
(a) Hierarchical Clustering With Less Feature Variables (b) Scree Plot With Less Feature Variables

Fig. 9: Hierarchical Clustering With Less Feature Variables Plots

The plots given in Fig. 9 are similar to the plots for the full data set given in Fig. 6 - the scree plot (see Fig. 9b) shows which number of clusters should be used in further analysis.

Using hierarchical clustering, the following accouracies are derived:

- 2 clusters: 58.54%
- 6 clusters: 82.93%
- 10 clusters: 82.93%
- 15 clusters: 90.24%

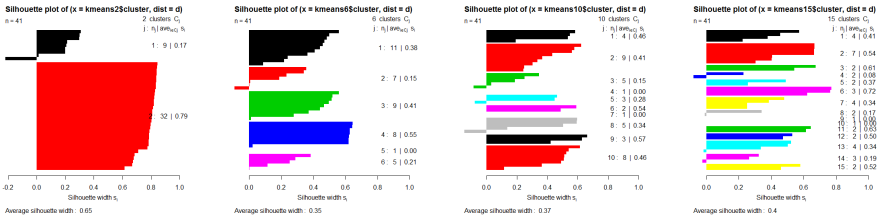


(a) 2 Clusters (b) 6 Clusters (c) 10 Clusters (d) 15 Clusters

Fig. 10: Silhouette Plot For Different Numbers of Clusters Using Hierarchical Clustering With Less Feature Variables

For k-means clustering the result is as follows:

- 2 clusters: 75.61%
- 6 clusters: 80.49%
- 10 clusters: 80.49%
- 15 clusters: 87.80%



(a) 2 Clusters (b) 6 Clusters (c) 10 Clusters (d) 15 Clusters

Fig. 11: Silhouette Plot For Different Numbers of Clusters Using K-Means Clustering With Less Feature Variables

The accuracies achieved with a lower number of features are slightly lower than the ones achieved with a higher number of features – but they are still not

too bad. Probably a number of features between the two would be optimal but a more in-depth analysis only seems feasible if an analysis of the effect the number of logs has on the results (i.e. does the approach even work for a larger number of logs) has been done.

3.8 Application to Log Data - Scenario 2

To start working with the second data set it has to be determined which of the parameters can be used for representing a log and ultimately be used in classification. Moreover, the logs have to be checked in order to find out which of them might not be appropriate for usage because they are somehow damaged or simply too short.

First off, it is analysed how many data points are contained in each log. Afterwards the decision that only logs which contain more than 100 data points are feasible to use is established. Therefore, 3 of the 205 logs are not taken into account for the further steps meaning that 202 logs can still be used for the analysis. Using the 202 remaining logs, all feature variables which are present in all of them are determined. From these 20 parameters the minimum number of occurrence within the 202 valid logs is determined. A threshold of 10 is chosen and all of the feature variables which are below that are deleted, which leads to 14 parameters. The code used for this process is given at [8].

As there are about 70 parameters representing a log in the first scenario (10 last values of 7 IDs), which achieve feasible results, it is concluded that a similar number should be chosen. Therefore, the decision that 5 data points per parameter should represent a log (as $14 * 5$ equals 70) is taken. For the first scenario the last 10 data points of each log are used - for this scenario it is decided to use the last 5, 5 in the middle and the first 5 data points in order to represent logs as it is interesting which of these variants provides the best results also indicating which point in time of the manufacturing might be most significant for the part to be correct. The parameters used for representation are:

- ns=2;s=/Channel/MachineAxis/aaLeadP[u1,1]
- ns=2;s=/Channel/MachineAxis/aaLeadP[u1,2]
- ns=2;s=/Channel/MachineAxis/aaLeadP[u1,3]
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,1]
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,2]
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,3]
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,1]
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,2]
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,3]
- ns=2;s=/Channel/MachineAxis/aaVactB[u1,1]
- ns=2;s=/Channel/MachineAxis/aaVactB[u1,2]
- ns=2;s=/Channel/MachineAxis/aaVactB[u1,3]
- ns=2;s=/Channel/Spindle/actSpeed
- ns=2;s=/Channel/Spindle/driveLoad

The code for reading the data from the CSV files and preparing it for usage in R is given below and at [16]. It is already taken into account that the measurements are missing for 6 of the parts - these are not used for classification later on - therefore, 196 logs are remaining.

```
#-------- Daten einlesen

#install.packages("gdata")
#install.packages("cluster")
#install.packages("clue")
#install.packages("rlist")
library(gdata)
library(cluster)
library(clue)
library(rlist)

setwd("P:\\Daten\\Uni\\Master Wirtschaftsinformatik\\2018 WS\\VU Business Intelligence II\\aufgabe4\\frage2")
#setwd("C:\\Users\\Admin\\Desktop\\bi2")

files<-list.files("machining")
files<-append(files[-c(1:10)],files[1:10])

alldata<-list()

for(i in 1:length(files)) {
  alldata[[i]]<-read.csv(paste("machining\\",files[i],sep=""),header=TRUE,sep="*",stringsAsFactors=FALSE)
}

allsplitdata<-list()
for(i in c(1:length(files))) {
  dataset<-alldata[[i]]
  ordereddata<-dataset[order(dataset$Id,dataset$ServerTimestamp,dataset$timestamp),]
  splitdata<-split(ordereddata,ordereddata$Id)
  allsplitdata[[i]]<-splitdata
}

dataitems<-array()
for(i in 1:length(files)) {
  dataitems[i]<-nrow(alldata[i][[1]])
}

goodfiles<-c()
for(i in 1:length(files)) {
  if(dataitems[i]>=100) {
    goodfiles<-c(goodfiles,i)
  }
}

listofids<-c()
for(i in goodfiles) {
  listofids<-c(listofids,names(allsplitdata[[i]]))
}

ids=unique(listofids)

sumofids<-array()
for(i in c(1:length(ids))) {
  counter<-0
  for(j in c(1:length(listofids))) {
    if(ids[i]==listofids[j]) counter<-counter+1
  }
  sumofids[i]<-counter
}

```

```
60 idswithsums<-cbind(ids,sumofids)
61
62 goodids<-c()
63 for(i in c(1:length(ids))) {
64   if(idswithsums[i,2]=="202") goodids<-c(goodids,idswithsums[i,1])
65 }
66
67 numbers<-data.frame()
68 for(i in goodfiles) {
69   buffer<-array()
70   buffer[1]<-i
71   for(j in c(1:length(goodids))) {
72     splitid<-which(goodids[j] == names(allsplitdata[[i]]))
73   buffer[j+1]<-nrow(allsplitdata[[i]][[splitid]])
74   }
75   numbers<-rbind(numbers,buffer)
76 }
77 numbers<-setNames(numbers,c("filenumber",goodids))
78
79 minimum<-array()
80 for(i in 2:21) {
81   minimum[i-1]<-min(numbers[i])
82 }
83
84 idswithminimum<-cbind(goodids,minimum)
85
86 reallygoodids<-c()
87 for(i in c(1:length(goodids))) {
88   if(as.numeric(idswithminimum[i,2])>=10) reallygoodids<-c(
       reallygoodids,idswithminimum[i,1])
89 }
90
91 gooddata<-data.frame();
92 for(i in goodfiles) {
93   buffer<-array()
94   buffer1<-array()
95   buffer2<-array()
96   for(j in c(1:length(reallygoodids))) {
97     splitid<-which(reallygoodids[j] == names(allsplitdata[[i]]))
98   #for(k in c(1:5)) {
99   #  buffer[(j-1)*5+k]<-as.numeric(allsplitdata[[i]][[splitid]]$value[
       k])
100  #}
101  #for(k in c(1:5)) {
102  #  buffer[(j-1)*5+k]<-as.numeric(allsplitdata[[i]][[splitid]]$value[
       round(nrow(allsplitdata[[i]][[splitid]])/2)-3+k])
103  #}
104  for(k in c(1:5)) {
105    buffer[(j-1)*5+k]<-as.numeric(allsplitdata[[i]][[splitid]]$value[
         nrow(allsplitdata[[i]][[splitid]])-(5-k)])
106  }
107  }
108  gooddata<-rbind(gooddata,buffer)
109 }
110 gooddatanames=c()
111 for(i in c(1:length(reallygoodids))) {
112   for(j in c(1:5)) {
113     gooddatanames=c(gooddatanames,paste(reallygoodids[i],j,sep=""))
114   }
115 }
116 gooddata<-setNames(gooddata,gooddatanames)
117
118 successdata<-read.csv("measuring.csv",header=TRUE,sep="*",
       stringsAsFactors=FALSE)
119 #successdataold<-successdata
120
121 helperMM1<-successdata$MM1[179]
122 helperMM2<-successdata$MM2[179]
```

```
123 helperMM3<-successdata$MM3[179]
124
125 helper1MM1<-successdata$MM1[180]
126 helper1MM2<-successdata$MM2[180]
127 helper1MM3<-successdata$MM3[180]
128
129 for(i in c(180:129)) {
130   successdata$MM1[i]<-successdata$MM1[i-2]
131   successdata$MM2[i]<-successdata$MM2[i-2]
132   successdata$MM3[i]<-successdata$MM3[i-2]
133 }
134
135 successdata$MM1[127]<-helperMM1
136 successdata$MM2[127]<-helperMM2
137 successdata$MM3[127]<-helperMM3
138
139 successdata$MM1[128]<-helper1MM1
140 successdata$MM2[128]<-helper1MM2
141 successdata$MM3[128]<-helper1MM3
142
143
144 goodsuccessdata<-data.frame()
145 for(i in goodfiles) {
146   goodsuccessdata<-rbind(goodsuccessdata,successdata[i,c(3:17)])
147 }
148 gooddata<-cbind(gooddata,goodsuccessdata)
149
150
151 j<-1
152 badgooddata<-array()
153 for(i in c(nrow(gooddata)):1) {
154   i<-i
155   if(is.na(gooddata$MM1[i]) || gooddata$MM1[i]=="") {
156     badgooddata[j]<-i
157     j<-j+1
158   }
159 }
160
161
162 for(i in badgooddata) {
163   gooddata<-gooddata[-i,]
164 }
165
166
167 for(i in c(71:85)) {
168   gooddata[,c(i)]<-(as.logical(gooddata[,c(i)]))
169 }
170
171 gooddata<-cbind(gooddata, acceptance=rowSums(gooddata[,77:85])==9)
172
173 gooddata<-cbind(gooddata, acceptance_MM=rowSums(gooddata[,71:73])==3)
```

As described above, different quintetts of data points are used to represent the logs. The difference between using the last, first, or middle five points lies in the following lines in the code:

For using the first five data points per parameter for representation the following code is used (available at [15]):

```
1 for(i in goodfiles) {
2   buffer<-array()
3   buffer1<-array()
4   buffer2<-array()
5   for(j in c(1:length(reallygoodids))) {
6     splitid<-which(reallygoodids[j] == names(allsplitdata[[i]]))
7   for(k in c(1:5)) {
```

```
8       buffer[(j-1)*5+k]<-as.numeric(allsplitdata[[i]][[splitid]]$value[k
            ])
9     }
10    #for(k in c(1:5)) {
11    #   buffer[(j-1)*5+k]<-as.numeric(allsplitdata[[i]][[splitid]]$value[
          round(nrow(allsplitdata[[i]][[splitid]])/2)-3+k])
12    #}
13    #for(k in c(1:5)) {
14    #   buffer[(j-1)*5+k]<-as.numeric(allsplitdata[[i]][[splitid]]$value[
          nrow(allsplitdata[[i]][[splitid]])-(5-k)])
15    #}
16    }
17    gooddata<-rbind(gooddata,buffer)
18  }
```

For using the middle five data points per parameter for representation the following code is used (available at [17]):

```
1  for(i in goodfiles) {
2    buffer<-array()
3    buffer1<-array()
4    buffer2<-array()
5    for(j in c(1:length(reallygoodids))) {
6      splitid<-which(reallygoodids[j] == names(allsplitdata[[i]]))
7    #for(k in c(1:5)) {
8    #   buffer[(j-1)*5+k]<-as.numeric(allsplitdata[[i]][[splitid]]$value[
          k])
9    #}
10   for(k in c(1:5)) {
11     buffer[(j-1)*5+k]<-as.numeric(allsplitdata[[i]][[splitid]]$value[
           round(nrow(allsplitdata[[i]][[splitid]])/2)-3+k])
12   }
13   #for(k in c(1:5)) {
14   #   buffer[(j-1)*5+k]<-as.numeric(allsplitdata[[i]][[splitid]]$value[
          nrow(allsplitdata[[i]][[splitid]])-(5-k)])
15   #}
16   }
17   gooddata<-rbind(gooddata,buffer)
18 }
```

Looking at the measurements/tests it is noteworthy that some of them are passed/not passed every time (or close to every time) which makes them difficult to use for classification because predicting the class which occurs very often provides a good result without having to rely on actual data.

The chart in Fig. 12 shows the number of parts passing the test for each test - the red line indicates the number of parts produced so the number of parts not passing the test can also be determined. As described above, measurements which have a good ratio of “passed” to “not passed” should be used for the classification - therefore, “Kreis.19.2.1.Konzentrizitaet”, “Kreis.19.2.2.Konzentrizitaet” and “Zylinder.4.5.B.Durchmesser” seem good to use. In order to also have a manual measurement in the classification the best one of these (“MM1”) is also added. The two bars on the right side represent the overall acceptance for automatic (all but the first 3 automatic measurements are OK - i.e. all but the ones starting with “Flaeche”) and manual measurements (all three manual measurements are OK).

The goal is to find out if the result of a single measurement can be predicted via classification. For the classification the last 5 data points of each machining log are used. Using the last few data points worked well for the first scenario. Using the last 5 data points of each feature variable gives 70 variables in total

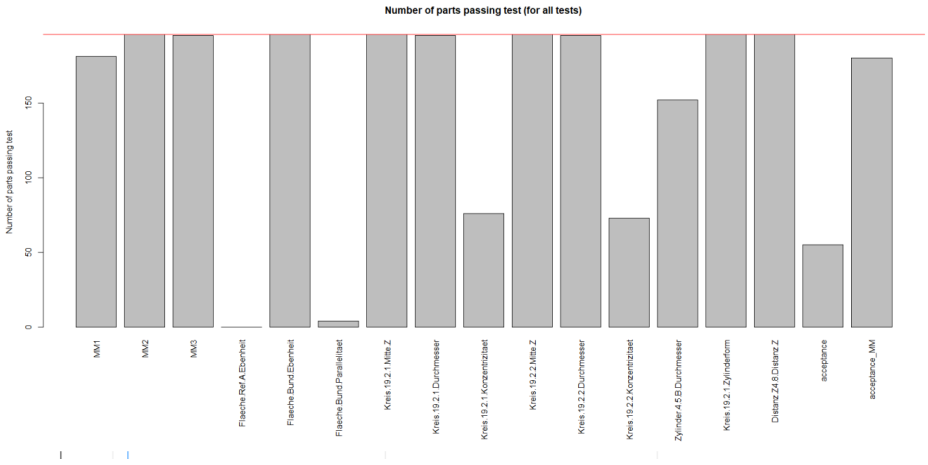


Fig. 12: Number of Parts Passing Test (For All Tests)

and 196 usable data points for each of those variables, which is a decent ratio. Measurements are chosen, where there is a good ratio between true and false (ok and nok), since trying to predict a measurement result, which is 99% ok, is probably rather pointless. Therefore, 1 manual measurement and 3 automatic measurements are chosen, which have a good ratio. Those measurements are, including their ratio of true/false:

- Manual Measurement 1 (181 ok/true(92,3%), 15 nok/false(7,7%))
- Kreis 19,2-1 Konzentrizitaet (76 ok/true(38,8%), 120 nok/false(61,2%))
- Kreis 19,2-2 Konzentrizitaet (73 ok/true(37,2%), 123 nok/false(62,8%))
- Zylinder 4,5-B Durchmesser (152 ok/true(77,6%), 44 nok/false(22,4%))

Although manual measurement 1 has a really high number of measurements with the result ok, it is still the lowest of all manual measurements, and it is useful to look at classification results for one manual measurement as well, to see if it behaves differently.

The data was split into a training set (75%, 143 logs) and a test set (25%, 53 logs).

For classification different algorithms were used, to give an idea about the accuracy of classification and if the models could be viable for classifying the logs. The code used for this analysis as well as the R output can be found at [9].

Classification was first performed for "ManualMeasurement1" using the last 5 data points (Tab. 2). The accuracy for the SVM with linear kernel with 97,9% for the training set and 84,9% for the test set seems very good, but looking at the predictions, which all predict only one class and the initial ratio of true/false, it can be seen that the classification is indeed very bad. The accuracies in the testsets for other kernels with 94,3% for polynomial and 96,2% for radial and sigmoid also seem really good, but a closer look shows, that the SVM figured out

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	0.01	1	1	-
gamma	0.01428571	0.01428571	0.001	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	52	91	31	32	83	-
accuracy training set	97.90%	91.61%	90.91%	90.91%	93.71%	72.73%
accuracy test set	84.91%	96.23%	96.23%	96.23%	94.34%	54.72%

Tab. 2: Classification Results For "ManualMeasurement1" Using The Last 5 Data Points

to just predict everything true, therefore getting a really high accuracy, but is basically not useful for actual predicting. Naive Bayes doesn't work at all here.

Those high accuracies can be attributed to the fact that the initial ratio of manual measurement 1 is really skewed in favour of true/ok. Therefore classification doesn't seem viable here. In the next step classification is used for the automatic measurements, which have a far more balanced ratio between true/ok and false/nok and should, if classification is viable, provide better results.

For classification for the measurement results of Kreis 19,2-1 Konzentrizitaet, Kreis 19,2-2 Konzentrizitaet and Zylinder 4,5-B Durchmesser the same steps as above for Manual Measurement 1 are used, therefore just the results of the different SVM kernels and naive bayes are presented in the Tab. 3, 4, and 5.

The results for Kreis19,2-1 Konzentrizitaet (Tab. 3) show that the accuracies of the testsets of the the different classification methods range from 52,8% (Naive Bayes) to 66% (SVM with radial kernel). This is not a good accuracy at all and it seems like the classification does not really find a good way to predict the outcome. Apparently the SVM found its best way to predict, by predicting nearly everything as the label which has the majority of the dataset (the data set has about 62% labelled false). As can be seen for the "best" prediction, SVM with a radial kernel, which predicts 48 out of 53 labels as false. This suggests that classification is not really working for the goal stated. To further confirm this assumption, the classification using the last 5 data points for the other measurement results is performed.

The classification for Kreis 19,2-2 Konzentrizitaet (Tab. 4) shows that this is the same story as above. It predicts nearly everything as the label which is more common in the dataset, in this case false, to get the best result it can achieve.

The classification of Zylinder 4,5-B Durchmesser (Tab. 5) works even worse than the unsuccessful classifications from above, by just predicting everything as true, although only about 77% are labelled as such. Those results lead to the

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	1	1	1	-
gamma	0.01428571	0.01428571	0.1	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	103	129	143	117	132	-
accuracy training set	82.52%	77.62%	98.60%	58.74%	74.13%	66.43%
accuracy test set	56.60%	66.04%	66.04%	62.26%	62.26%	52.83%

Tab. 3: Classification Results For "Kreis19,2-1 Konzentrizitaet" Using The Last 5 Data Points

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	1	1	1	-
gamma	0.01428571	0.01428571	0.1	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	104	130	143	110	130	-
accuracy training set	84.62%	73.43%	98.60%	58.04%	76.22%	66.43%
accuracy test set	52.83%	62.26%	62.26%	56.60%	60.38%	56.60%

Tab. 4: Classification Results For "Kreis19,2-2 Konzentrizitaet" Using The Last 5 Data Points

conclusion that classification for this data just does not work. One possibility could be, that the last 5 data points are just not suitable for this case. Therefore, classification is again performed for the same measurement results, but with the first 5 data points of the machining data.

For classification for all the measurement results using the first 5 data points the same steps as above for manual measurement 1 using the last 5 data points are carried out. The only difference is the different data set, which was used as source, which contains the first 5 data points per parameter to describe a log.

Again the results given in Tab. 6, 7, 8, and, 9 indicate that, just like using the last 5 data points, all the classifications just predict everything as true or false,

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	0.01	1	1	-
gamma	0.01428571	0.01428571	0.001	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	76	114	67	70	111	-
accuracy training set	88.11%	81.12%	78.32%	76.22%	87.41%	71.33%
accuracy test set	67.92%	75.47%	75.47%	77.36%	73.58%	60.38%

Tab. 5: Classification Results For "Zylinder 4,5-B Durchmesser" Using The Last 5 Data Points

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	0.01	1	1	-
gamma	0.01428571	0.01428571	0.001	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	40	84	33	35	104	-
accuracy training set	99.30%	93.01%	90.91%	90.21%	93.71%	33.57%
accuracy test set	83.02%	96.23%	96.23%	96.23%	96.23%	11.32%

Tab. 6: Classification Results For "ManualMeasurement1" Using The First 5 Data Points

depending which is the majority of the data set. It seems it doesn't even matter if the first or last 5 data points are used, which strongly suggests that there is just no correlation between the machining variables and the measurement result. Just to be sure, the classification for a relatively evenly distributed measurement result is repeated using the middle 5 data points of the machining data.

For classification for the measurement results of Kreis 19,2-1 Konzentrizitaet using the middle 5 data points the same steps as above for manual measurement 1 using the last 5 data points are executed. The only difference is the different data set, which is used as source, which contains the middle 5 data points per parameter to describe a log.

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	10	1	1	-
gamma	0.01428571	0.01428571	0.001	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	83	126	121	117	136	-
accuracy training set	88.81%	88.11%	82.52%	68.53%	69.93%	42.66%
accuracy test set	56.60%	62.26%	56.60%	60.38%	64.15%	37.74%

Tab. 7: Classification Results For "Kreis19,2-1 Konzentrizitaet" Using The First 5 Data Points

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	10	1	1	-
gamma	0.01428571	0.01428571	0.001	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	87	130	119	113	137	-
accuracy training set	88.81%	85.31%	82.52%	65.73%	72.03%	40.56%
accuracy test set	52.83%	60.38%	58.49%	60.38%	62.26%	39.62%

Tab. 8: Classification Results For "Kreis19,2-2 Konzentrizitaet" Using The First 5 Data Points

This classification (for results see Tab. 10) does again not seem to work and again predicts everything as false. This again confirms the assumption that classification is not really working out for the stated goal. To eliminate another potential problem, the number of feature variables is reduced, which potentially skew the classification. Here the method described in section 3.4 is used for the data sets using the last and first 5 data points for log representation.

The plot given in Fig. 13 suggests that more feature variables don't really lead to more accuracy at a certain point when using the last 5 data points to represent a log. Low amounts such as 14 feature variables already reach the same accuracy. Therefore, the 14 "best" features are used to create a new data frame and perform classification. The 14 feature variables are (the number after the

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	0.01	1	1	-
gamma	0.01428571	0.01428571	0.001	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	70	118	67	71	134	-
accuracy training set	96.50%	80.42%	78.32%	77.62%	81.12%	51.75%
accuracy test set	67.92%	75.47%	75.47%	75.47%	75.47%	49.06%

Tab. 9: Classification Results For "Zylinder 4,5-B Durchmesser" Using The First 5 Data Points

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	1	1	1	-
gamma	0.01428571	0.01428571	0.01	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	92	135	130	118	136	-
accuracy training set	88.11%	76.92%	71.33%	60.14%	72.73%	63.64%
accuracy test set	49.06%	66.04%	64.15%	64.15%	58.49%	52.83%

Tab. 10: Classification Results For "Kreis19,2-1 Konzentrizitaet" Using The Middle 5 Data Points

dot indicates which of the values within the 5 values chosen - this means that ".1" is the 5th value and ".5" is the first value counting from the last of the five values):

- ns=2;s=/Channel/MachineAxis/aaLoad[u1,2].1
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,2].2
- ns=2;s=/Channel/MachineAxis/aaLeadP[u1,1].5
- ns=2;s=/Channel/MachineAxis/aaLeadP[u1,1].4
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,2].4
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,2].3
- ns=2;s=/Channel/Spindle/driveLoad.3
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,3].5

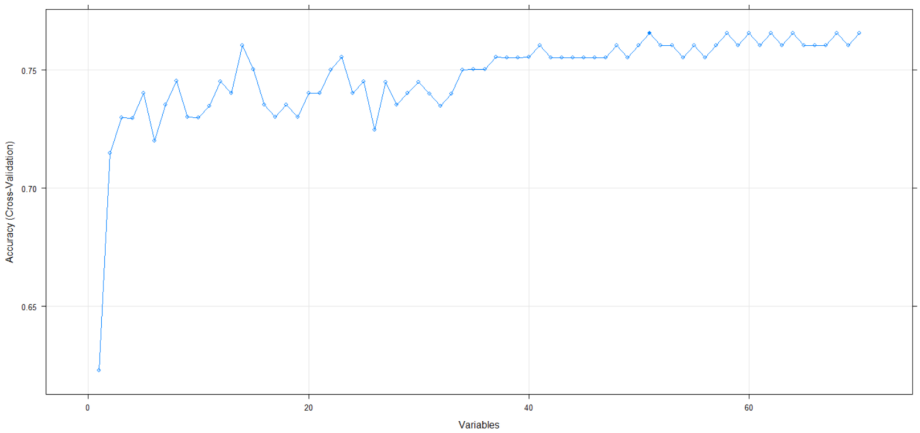


Fig. 13: Recursive Feature Elimination "Zylinder 4,5-B Durchmesser" Last 5 Data Points

- ns=2;s=/Channel/MachineAxis/aaTorque[u1,3].3
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,2].3
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,3].3
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,3].1
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,2].5
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,1].4

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	1	1	1	-
gamma	0.07142857	0.07142857	1	0.07142857	0.07142857	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	80	87	98	63	78	-
accuracy training set	78.32%	79.02%	87.41%	75.52%	82.52%	69.23%
accuracy test set	75.47%	75.47%	75.47%	75.47%	73.58%	60.38%

Tab. 11: Classification Results For "Zylinder 4,5-B Durchmesser" With Feature Elimination Using The Last 5 Data Points

Classification results given in Tab. 11 for “Zylinder 4,5-B Durchmesser” with feature elimination using the last 5 data points shows that even having less data points, nothing changed. Again the prediction is really one sided. To confirm that feature elimination doesn’t change the success of classification, feature elimination is also performed for the first 5 data points.

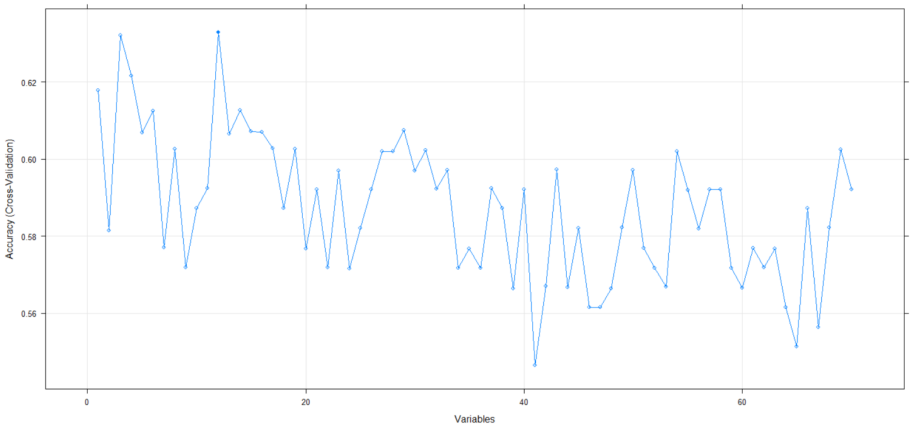


Fig. 14: Recursive Feature Elimination “Kreis19,2-2 Konzentrizitaet” First 5 Data Points

The plot given in Fig. 14 suggests that less feature variables lead to more accuracy up to a certain point when using the first 5 data points to represent a log. Low amounts such as 12 feature variables already reach the best accuracy. These 12 feature variables are (the number after the dot indicates which of the values within the 5 values chosen -this means that “.1” is the 5th value and “.5” is the first value counting from the last of the five values):

- ns=2;s=/Channel/MachineAxis/aaTorque[u1,1].5
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,2].3
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,3].5
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,1].2
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,2].1
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,1].4
- ns=2;s=/Channel/MachineAxis/aaLeadP[u1,1].4
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,1].3
- ns=2;s=/Channel/MachineAxis/aaLoad[u1,2].5
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,1].3
- ns=2;s=/Channel/MachineAxis/aaVactB[u1,3].5
- ns=2;s=/Channel/MachineAxis/aaTorque[u1,2].5

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	1	1	1	-
gamma	0.08333333	0.08333333	0.01	0.08333333	0.08333333	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	103	125	112	110	113	-
accuracy training set	69.23%	76.92%	64.34%	58.74%	73.43%	67.13%
accuracy test set	62.26%	56.60%	62.26%	62.26%	60.38%	67.92%

Tab. 12: Classification Results For "Kreis19,2-2 Konzentrizitaet" With Feature Elimination Using The First 5 Data Points

Removing feature variables in this approach leads to an interesting effect shown in Tab. 12. Instead of just going the safe route of predicting everything false, it predicts more wrongfully true, which naturally leads to a lower accuracy, than using all feature variables. This is the last attempt to predict the outcome of the measurement using only the machining data via classification. There could be certain data points which would work, but for this relatively generic way, the solution is not useful.

Since the first goal isn't achieved, a new approach is chosen, that looks if the machining data can predict the outcome of not only a single measurement, but the whole part. A part where the measurement of at least one automatic measurement, except the first three (the first three being the ones starting with "Flaeche" in Fig. 12), is NOK/FALSE is considered faulty. Again the same classification method as above is used. The classification is performed using the last 5, the first 5 and the middle 5 data points with naive bayes and different kernels for SVM.

Just like the classification for single measurement results, the SVM predicts everything as the majority of the labels (the test set is 75% false), which is just not a useable way for classification. Furthermore, no matter which data points are used, the results given in Tab. 13, 14, and 15 are very similar, which speaks for no correlation between the data points and the actual outcome of the part.

Another approach used is to look at manual measurements and automatic measurements and compare the results (as described above for the manual measurement to be OK/TRUE all three manual measurements have to be OK/TRUE and for the automatic measurement all but the first three need to be OK/TRUE).

As can be seen in Tab. 16, 12 of the 16 manual measurement results being NOK are actually NOK (with 4 being actually OK) and 51 of the 180 manual

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	0.01	1	1	-
gamma	0.01428571	0.01428571	0.001	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	89	125	88	91	121	-
accuracy training set	86.01%	77.62%	72.03%	70.63%	81.82%	70.63%
accuracy test set	64.15%	71.70%	71.70%	69.81%	67.92%	66.04%

Tab. 13: Classification Results For Overall Automatic Measurement Using The Last 5 Data Points

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	0.01	1	1	-
gamma	0.01428571	0.01428571	0.001	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	78	121	86	86	133	-
accuracy training set	90.91%	81.12%	72.03%	72.03%	78.32%	36.36%
accuracy test set	67.93%	71.70%	71.70%	69.81%	71.70%	32.08%

Tab. 14: Classification Results For Overall Automatic Measurement Using The First 5 Data Points

results being OK are actually OK (with 129 being actually NOK). This applies only under the assumption that the automatic measurement is always correct. These results are not good enough to predict the outcome of the actual (automatic measurement) result but it can definitely be used for some considerations. Thinking of a scenario where the automatic measurement is very expensive and the production of one part is quite cheap it might for example be a good idea to just make an automatic quality control for parts that pass the manual one and accept that with this approach some good parts might not be sold but nonetheless the overall revenue might be higher. More economic considerations with the goal to optimise the cash flow can be performed with further knowledge of the actual prices for measurements and production.

	SVM					Naive Bayes
kernel	linear	radial	radial (tuned)	sigmoid	polynomial	-
cost	1	1	0.01	1	1	-
gamma	0.01428571	0.01428571	0.001	0.01428571	0.01428571	-
degree	-	-	-	-	3	-
coef.0	-	-	-	0	0	-
# support vectors	83	131	89	91	121	-
accuracy training set	88.81%	77.62%	72.03%	72.03%	80.42%	64.34%
accuracy test set	60.38%	71.70%	71.70%	75.47%	67.92%	50.94%

Tab. 15: Classification Results For Overall Automatic Measurement Using The Middle 5 Data Points

	MM_true	MM_false
automatic_true	51	4
automatic_false	129	12

Tab. 16: Comparison Of Manual And Automatic Measurement

3.9 Lessons Learned

Overall, classification and clustering of the data achieves good results for the first scenario looking at it from the point of accuracy. A major problem with these results is that they seem unstable (e.g. looking at the different results for different seed values regarding the split between training and test data set) which is most likely based on the small number of logs available. To make a final conclusion on the precision of clustering and classification approaches using this data a higher number of logs would be needed. Additionally, there is the problem of not all logs measuring the same variables. If this was more standardized, it would be possible to use a higher number of variables for analysis which would probably make the results even better. All in all, the results obtained meet the goal of describing the data given pretty well but the question is if this result is meaningful as this would require a higher number of logs.

For, the second scenario classification does not provide satisfactory results looking at the goal of using the machining data to predict measurement results or the overall "OKness" of a part. There are quite a few reasons that could be made responsible for this lack of accuracy for the classification methods all having as root cause that the wrong machining data is used for prediction with the most obvious one being that 5 data points per parameter do simply not contain enough information to make a meaningful prediction. Another reason

could be that due to trying it with only 3 different quintetts (first 5, middle 5, last 5) the region that is important for the quality of the part is not covered with the data selected. Furthermore, it is important to also take into consideration that it might not be possible to predict product quality based on the machining data provided at all.

A further difficulty in performing classification is the ratio of passed to not passed tests for most of the measurements - as discussed in the beginning of section 3.8, having a very high percentage of parts failing or succeeding at a test makes it impossible to obtain a feasible prediction formula as just predicting the result which occurs in nearly all cases provides a very high accuracy.

Another interesting point which is discussed in greater detail in the last paragraphs of section 3.8 is the role manual measuring has when it comes to determining the quality of parts. Even if manual measuring does not really give a hint about how the result of the automatic measurement will be, it could be useful for taking economic decisions. It might for example be feasible to decide if the automatic measurements should be carried out based on the manual measurement. The outcome of this consideration depends strongly on the cost of manufacturing per part, the cost of automatic measurements as well as of the costs that are anticipated for selling faulty parts.

4 Conclusion and Outlook

This work shows how conformance checking, classification, and clustering of machining data gathered by executing a BPMN manufacturing process with a workflow engine can be performed. Furthermore, the results achieved are shown and discussed with respect to their quality, possible improvements, and suggestions for future work. Overall, the results show that data collected within a manufacturing process can be used for process-oriented analysis such as conformance checking of process logs or classification and clustering of the processes based on machining data in order to predict if produced parts are good or bad. In contrast to the resource-based data collection method where data-streams of single machines have to be saved in databases, cleaned and re-contextualized by connecting it to orders, batches, or single products, the aforementioned method has the advantage of providing data for the whole BPMN process which makes it easy to retrieve data for individual parts for further analysis. Future work in this field will include visualizing the data contained within the process logs, finding standardized ways to represent logs of individual parts to make it possible to compare the machining data points of different parts, and determining ways to choose parameters that should be recorded during the process execution. Additionally, it would be desirable to evaluate whether the ex-post analysis performed in this paper can also be undertaken at run-time.

Acknowledgements: This work has been partially supported and funded by the Austrian Research Promotion Agency (FFG) via the “Austrian Competence Center for Digital Production” (CDP) under the contract number 854187.

References

1. Adriansyah, A.: Replay a log on petri net for conformance analysis plug-in. https://svn.win.tue.nl/trac/prom/export/41469/Documentation/ReplayerPackageDocumentation/Replay%20a%20Log%20on%20Petri%20Net%20for%20Conformance%20Analysis.pdf (2011), [online; accessed 25-February-2019]
2. Adriansyah, A.: Replayerpackagedocumentation. https://svn.win.tue.nl/trac/prom/browser/Documentation/ReplayerPackageDocumentation/ (2011), [online; accessed 25-February-2019]
3. Bermingham, M.L., Pong-Wong, R., Spiliopoulou, A., Hayward, C., Rudan, I., Campbell, H., Wright, A.F., Wilson, J.F., Agakov, F., Navarro, P., et al.: Application of high-dimensional feature selection: evaluation for genomic prediction in man. Scientific reports 5, 10312 (2015)
4. Ho, T.K.: Random decision forests. In: Proceedings of 3rd international conference on document analysis and recognition. vol. 1, pp. 278–282. IEEE (1995)
5. Jonathan Sumrall, Juergen-Albrecht Fassmann, M.E.: xes.py. http://cpee.org/~demo/bus_paper/code/conf/xes.py, [online; accessed 25-February-2019]
6. Kuhn, M.: Recursive feature elimination. https://topepo.github.io/caret/recursive-feature-elimination.html (2018), [online; accessed 26-February-2019]
7. Mangler, J., Stuermer, G., Schikuta, E.: Cloud process execution engine - evaluation of the core concepts. CoRR abs/1003.3330 (2010), http://arxiv.org/abs/1003.3330
8. Matthias Ehrendorfer, J.A.F.: analysis.py - scenario 2. http://cpee.org/~demo/bus_paper/code/cluss/scenario2/analysis.txt, [online; accessed 25-February-2019]
9. Matthias Ehrendorfer, J.A.F.: ausgabe_r.txt - scenario 2. http://cpee.org/~demo/bus_paper/code/cluss/scenario2/ausgabe_r.txt, [online; accessed 25-February-2019]
10. Matthias Ehrendorfer, J.A.F.: classification.txt - scenario 1. http://cpee.org/~demo/bus_paper/code/cluss/scenario1/classification.txt, [online; accessed 25-February-2019]
11. Matthias Ehrendorfer, J.A.F.: clusteringfulldata.txt - scenario 1. http://cpee.org/~demo/bus_paper/code/cluss/scenario1/clusteringFullData.txt, [online; accessed 25-February-2019]
12. Matthias Ehrendorfer, J.A.F.: clusteringlessdata.txt - scenario 1. http://cpee.org/~demo/bus_paper/code/cluss/scenario1/clustering_lessData.txt, [online; accessed 25-February-2019]
13. Matthias Ehrendorfer, J.A.F.: csv_map.py - scenario 1. http://cpee.org/~demo/bus_paper/code/cluss/scenario1/csv_map.py, [online; accessed 25-February-2019]
14. Matthias Ehrendorfer, J.A.F.: csv_map.py - scenario 2. http://cpee.org/~demo/bus_paper/code/cluss/scenario2/csv_map.py, [online; accessed 25-February-2019]
15. Matthias Ehrendorfer, J.A.F.: daten_einlesen_anfang.txt - scenario 2. http://cpee.org/~demo/bus_paper/code/cluss/scenario2/daten_einlesen_anfang.txt, [online; accessed 25-February-2019]
16. Matthias Ehrendorfer, J.A.F.: daten_einlesen_ende.txt - scenario 2. http://cpee.org/~demo/bus_paper/code/cluss/scenario2/daten_einlesen_ende.txt, [online; accessed 25-February-2019]

17. Matthias Ehrendorfer, J.A.F.: daten_einlesen_mitte.txt - scenario 2. http://cpee.org/~demo/bus_paper/code/cluss/scenario2/daten_einlesen_mitte.txt, [online; accessed 25-February-2019]
18. Matthias Ehrendorfer, J.A.F.: filepaths.txt - scenario 1. http://cpee.org/~demo/bus_paper/code/conf/scenario1/filepaths.txt, [online; accessed 25-February-2019]
19. Matthias Ehrendorfer, J.A.F.: filepaths.txt - scenario 2. http://cpee.org/~demo/bus_paper/code/conf/scenario2/filepaths.txt, [online; accessed 25-February-2019]
20. Matthias Ehrendorfer, J.A.F.: machining_filepaths.txt - scenario 1. http://cpee.org/~demo/bus_paper/code/cluss/scenario1/filepaths_machining.txt, [online; accessed 25-February-2019]
21. Matthias Ehrendorfer, J.A.F.: machining_filepaths.txt - scenario 2. http://cpee.org/~demo/bus_paper/code/cluss/scenario2/machiningFilepaths.txt, [online; accessed 25-February-2019]
22. Matthias Ehrendorfer, J.A.F.: measuring.py - scenario 2. http://cpee.org/~demo/bus_paper/code/cluss/scenario2/measuring.py, [online; accessed 25-February-2019]
23. Matthias Ehrendorfer, J.A.F.: processtotpn.py - scenario 1. http://cpee.org/~demo/bus_paper/code/conf/scenario1/processToTpn.py, [online; accessed 25-February-2019]
24. Matthias Ehrendorfer, J.A.F.: processtotpn.py - scenario 2. http://cpee.org/~demo/bus_paper/code/conf/scenario2/processToTpn.py, [online; accessed 25-February-2019]
25. Matthias Ehrendorfer, J.A.F.: templates.xlsx - scenario 1. http://cpee.org/~demo/bus_paper/supplements/conf/scenario1/templates, [online; accessed 25-February-2019]
26. Matthias Ehrendorfer, J.A.F.: xes_map.py - scenario 1. http://cpee.org/~demo/bus_paper/code/conf/scenario1/xes_map.py, [online; accessed 25-February-2019]
27. Matthias Ehrendorfer, J.A.F.: xes_map.py - scenario 2. http://cpee.org/~demo/bus_paper/code/conf/scenario2/xes_map.py, [online; accessed 25-February-2019]
28. Meyer, D.: e1071. https://www.rdocumentation.org/packages/e1071/versions/1.7-0.1 (2019), [online; accessed 13-March-2019]
29. Process Mining Group: Prom tools. http://promtools.org/doku.php (2010), [online; accessed 25-February-2019]