同济大学 TONGJI UNIVERSITY
Chinesisch-Deutsche Hochschule für Angewandte Wissenschaften der Tongji-Universität (CDHAW)
中德工程学院

Mechatronics Project Report

Machine Learning for Industry 4.0

Lukas Froebus, Georg Hammer, Lucky Iheme
2018, CDHAW, TONGJI UNIVERSITY, Huang Ha

Content

Introduction

The Mechatronics Project at the Sino German College of Applied Engineering of Tongji University is a student project to practically apply learned knowledge. Improving the understanding about Machine Learning is the main objective of the project Machine Learning for Industry 4.0.

This Report will give an overview of the project, the approach to solve the problem and the results that were obtained.

Machine Learning offers great potential in the industrial environment as it is capable to analyse large datasets automatically. Machines can learn to optimize themselves and improve their quality.

Internet of Things

"The Internet of Things (IoT), Industry 4.0 or Smart production – even if the name varies it is always about the seamless connection of the digital and the real worlds."

The above-mentioned company's vision of Industry 4.0 summarizes the main aspect of Industry 4.0 clearly. The ongoing digitalization, which was triggered by major advancements in information technology, will soon connect any device or sensor to the global network. Industry 4.0 is usually referred to the current trend of the increasing data-exchange in manufacturing technologies.

Since the introduction of mass production in the course of the second industrial revolution, the demand for customization of products has become the greatest challenge for the manufacturing sector nowadays. The need for individualization and therefore the shrinkage of batch sizes has caused a completely new competitive environment for all manufacturers. This fast-changing environment induce the development from a static mass production to a highly flexible (mass-) production.

By connecting machines, work pieces and systems, businesses are creating intelligent networks along their entire value chain that can control each other autonomously. Furthermore, data is acquired throughout the whole product life cycle, even during operations. Therefore, the world of manufacturing will become increasingly connected until everything is linked to each other. This means that the boundaries of individual factories will no longer exists, and the complexity of production and supplier networks will grow enormously.

There are several differences between traditional factories and Industry 4.0 factories. While traditional factories and smart factories both targeting to provide high-quality products with the

least cost what is known as the maximization of performance, various data sources are available to supply valuable information about any production step. In traditional manufacturing systems, this data is used for understanding current operation stages and detecting faults and failures. In Industry 4.0 scenarios a new worthwhile level is added to the continuous operation monitoring and failure observation, it is that the components and machines gain self-awareness, self-predictiveness, self-adjustment and the ability to communicate.

Challenges

IT security issues due to the opening of the former closed factory network to the global network must be considered carefully. Whoever controls the company's network, also controls the whole manufacturing system. Therefore, it is highly recommended to firstly ensure a powerful network protection throughout all applied IT systems. Furthermore, the employees may be taken into responsibility for cyber security. USB-drives from unknown sources or poor passwords are still one of the greatest dangers for the whole network. The worst-case scenario, a complete network shutdown, must be avoided in any case as those would cause an expensive production outage.

Any control file of industrial automation gear holds a lot of industrial knowledge. Consequently, it is obligatory to encrypt the data uploaded to the network. Otherwise, crucial process or product knowledge is easy to be stolen during cyber-attacks or by employees taking part in the production process.

The increasing demand for machine learning applications, drives the need for more engineers specializing in the field of data science and software engineering

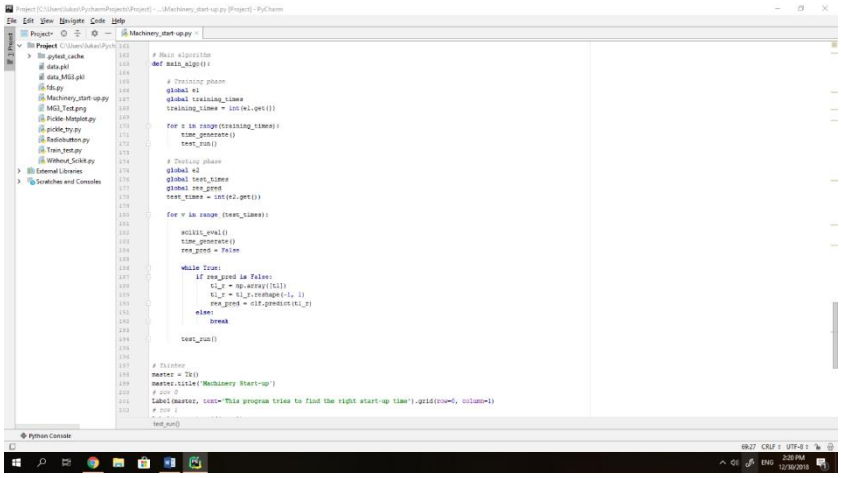
Task Description

During a manual start operation an unlock button must be pressed followed by a start button. These two buttons must be pressed in a certain order, with an unknown waiting period between. The project must deliver a working waiting period and incorporate a machine learning function.

Software

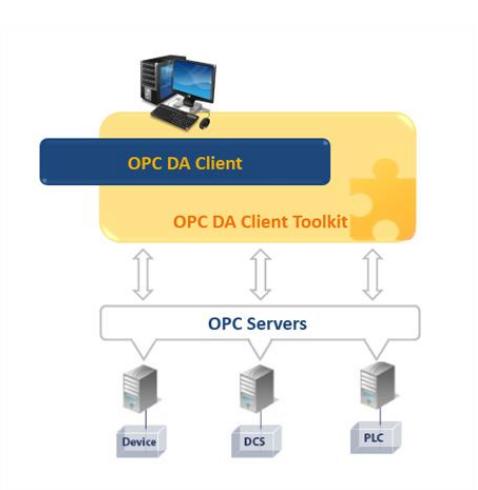
Since Python is perfectly adopted for the implementation of machine learning algorithms, it was chosen as the go-to language for our project. Especially, most open-source modules for machine-learning applications, make it to the optimal language. PyCharm is a well-known development environment for Python.

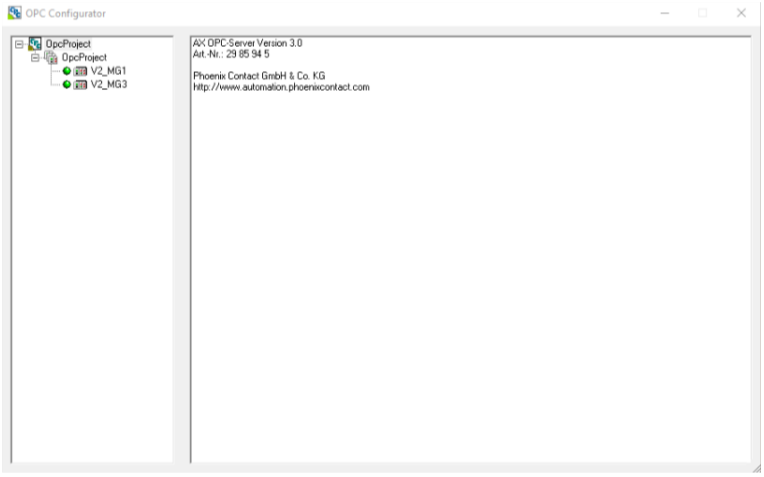
To establish a valid connection to the manufacturing line, it is necessary to wire the controlling computer to one of the ethernet nodes. Due to some experienced problems, it was necessary to set up the IP-address manually in some cases.



OpenOPC

OPC is a standard for secure and reliable exchange of data in industrial automation. The platform ensures the seamless flow of information among devices from multiple vendors. OPC is a



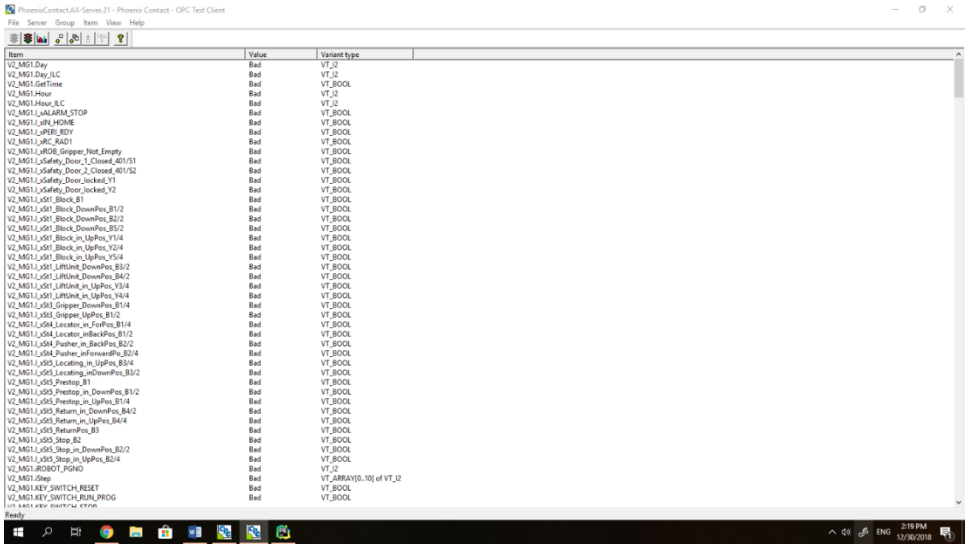


standard for secure and reliable exchange of data in industrial automation. The platform ensures the seamless flow of information among devices from multiple vendors. The OPC standard is a

series of specifications developed by industry vendors, end-users and software developers. These specifications define the interface between Clients and Servers, as well as Servers and Servers, including access to real-time data, monitoring of alarms and events, access to historical data and other applications. In this case, OpenOPC.

The software of Phoenix Contact consists of two components, the first component “OPC Configurator” is used to configure the client, respectively to point the client to the desired OPC

server by entering the IP-address. The second part of the software is the actual client, which can display or to write values to the connected OPC-server.

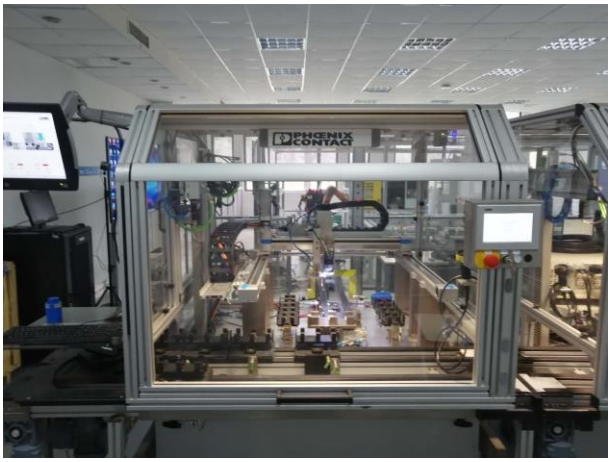


Python

This project utilizes the OPC-DA (Win32 COM based) industrial automation standard. Due to this it is obligatory to use a Python 32-bit version. Another requirement by the used OpenOPC library version 1.3.1 for Python is the usage of a Python version 2.7.xx. If any of the above-mentioned requirements is not satisfied, no valid connection can be set-up resulting in a bunch of failures.

Hardware

The test runs are carried out on Machine 1 and Machine 3 of the Industry 4.0 Laboratory. Pictured below.



Machine 1

Machine 3

Functions

Below are various parts of the Code, Functions and their corresponding explanation.

Import

```
import OpenOPC
import numpy as np
import cPickle as pickle
import time
from Tkinter import *
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

Import loads new Modules into the Python File. These Modules are necessary for the Code to run and have all the necessary functionality.

OpenOPC allows to use OPC commands and therefore establish a connection to the machine. Numpy is Pythons math module and offers enhanced mathematical capabilities including a random function. The additional mathematical capabilities are useful during the code to manipulate arrays and manage different calculations. To simplify the coding process, it's imported as "np", "np" being the abbreviation used for numpy. With cPickle we can include pickle functionality which is a database storage function in python. Again, an abbreviation is used to reduce complexity with "pickle". The time module allows us to add real-time functionality to our program. This is necessary to adjust the programs time. Pythons Tkinter Module is necessary to create working graphical user interfaces. The sklearn modules ads machine learning capabilities to the python program, classifiers, metrics and training sets are imported. Lastly pyplot from the matplotlib library is imported to allow the creation of plots.

Connection to PLC

```
opc = OpenOPC.client()
opc.connect('PhoenixContact.AX-Server.21')
time.sleep(2)
```

An instance of the opc client is introduced with "opc", afterwards a connection is established to the Phoenix Contact OPC Server. To ensure there is enough time to establish the connection correctly the script waits two seconds to give time to complete the operation. After this the server can simply be addressed as "opc" and additional commands to interact with the Server can be used.

Database creation

```
def create_new_pickle():
    with open(pickle_name, 'wb') as pickle_file:
        pickle.dump([[], []], pickle_file)
```

This function “create_new_pickle” opens a new empty pickle database and creates empty arrays to save the collected data in it. The file is named “data.pkl” and allows binary writing.

Saving Results in Database

```
def ins_pickle():
    global current_p_list

    while True:
        try:
            with open(pickle_name, 'rb') as pickle_file:
                current_p_list = pickle.load(pickle_file)

            for jj in range(len(times_current_run)):
                current_p_list[0].append(times_current_run[jj])
                current_p_list[1].append(results_current_run[jj])

            with open(pickle_name, 'wb') as pickle_file:
                pickle.dump(current_p_list, pickle_file)

            break

        except IOError:
            create_new_pickle()
```

Due to several test runs, it is known that the run of the program apparently fails, whenever no pickle file with the defined name was found. Because of this, the export function contains a “try and catch” statement with an infinite loop. The program escapes the infinite loop when all the values have been successfully written to the pickle file. For this, the program opens the existing database and copies it content on two temporary arrays. Those arrays are appended with the time variables and their belonging result. Finally, the former pickle file is overwritten with the new content of the temporary arrays. In the other case, the program starts the function “create_new_pickle” which creates a new pickle file.

Main Function

```
def test_run():
    global times_current_run
    global results_current_run
    times_current_run = []
    results_current_run = []
```

```
        times_current_run.append(t1)
        opc.write([machine_unlock, True])
        time.sleep(t1)
        opc.write([machine_on, True])
        time.sleep(7.5)
        result = opc[machine_check]
        print(result)
        results_current_run.append(result)
        opc.write([machine_off, True])
        time.sleep(7.5)

        ins_pickle()
```

At the beginning of the “test_run” function several global and local variables are declared and the function to generate a random time is being executed and the results is added to an array. In this function the main exercise of the project is done. The program virtually presses the unlock-button and after that the start button with the defined delay. It is necessary to make sure that the machine had enough time to start-up. Therefore, the check is executed after a delay of 5 seconds. Finally, the result is written to another array, the machine is shut down and the pickle file is being written.

Scikit

```
def scikit_eval():
    global clf
    global table
    table = []

    with open(pickle_name, 'rb') as pickle_file:
        table = pickle.load(pickle_file)
        x = table[0]
        y = table[1]
    # n_neighbors refers to the number of neighbors to be tested
    clf = KNeighborsClassifier(n_neighbors=2)
    # Generally execute Train_Test_Split by using 20 to 40% for testing
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=4)
    # Switch type from list to array
    x_train_a = np.asarray(x_train)
    x_test_a = np.asarray(x_test)
    y_train_a = np.asarray(y_train)
    y_test_a = np.asarray(y_test)
    # transpose the features matrix
    x_train_ar = x_train_a.reshape(-1, 1)
    x_test_ar = x_test_a.reshape(-1, 1)
    # Training the machine with the training data
    clf.fit(x_train_ar, y_train_a)
    # Testing the machine with the test data
    y_pred = clf.predict(x_test_ar)
    # Printing the result
    accuracy = metrics.accuracy_score(y_test_a, y_pred)
    print('The trained model has an accuracy of: ' + str(np.round(accuracy, 6)
* 100) + '%')
```

This part of the code implements the supervised machine learning function. For this machine learning algorithm, the scikit learn library is imported to the project. The exercise to find a specific result depending on several features is known as a classical classification problem. This problem is solved by the Nearest Neighbours classifier which predicts a specific outcome based on the five nearest neighbours. To ensure that the classifier is not following the noise of the outcome, it makes use of the train-test-split module. This module splits the database in two parts, one for training and another one for testing. This lets the program calculate the accuracy of the trained model. For raising the accuracy of the model, it is retrained after each run.

Main Algorithm

```
def main_algo():
    # machine selection
    global c
    global machine_unlock
    global machine_on
    global machine_check
    global machine_off
    global pickle_name
    global t1

    c = sel.get()
    if c == 1:
        machine_unlock = 'V2_MG1.Web_MachineUnlock_Button'
        machine_on = 'V2_MG1.Web_MachineOn_Button'
        machine_check = 'V2_MG1.O_xSt1_Motor_8_Anticlockwise'
        machine_off = 'V2_MG1.Web_MachineOff_Button'
        pickle_name = 'pickle_MG1.pkl'
        print(machine_unlock, machine_on, machine_check, machine_off)
    elif c == 3:
        machine_unlock = 'V2_MG3.Web_MachineUnlock_Button'
        machine_on = 'V2_MG3.Web_MachineOn_Button'
        machine_check = 'V2_MG3.OPC_xRobot_MotorOn_LED'
        machine_off = 'V2_MG3.Web_MachineOff_Button'
        pickle_name = 'pickle_MG3.pkl'
        print(machine_unlock, machine_on, machine_check, machine_off)
    else:
        print("ERROR")
    print("execution on Machine: ", c)

    # Training phase
    global e1
    global training_times
    training_times = int(e1.get())

    for z in range(training_times):
        t1 = round(np.random.uniform(2, 5), 4)
        test_run()

    # Testing phase
    global e2
```

```
    global test_times
    test_times = int(e2.get())

    scikit_eval()

    for v in range(test_times):

        while True:
            t1 = round(np.random.uniform(2, 5), 4)
            t1_r = np.array([t1])
            t1_r = t1_r.reshape(-1, 1)
            res_pred = clf.predict(t1_r)

            if res_pred == True:
                break

        test_run()
```

This code is executed after the “RUN” button is pressed. At the very beginning the Radio button selection is checked. Depending on the selection the code is executed either on Machine 1 or Machine 3. This configuration is made using an if statement. Specific values are stored to unlock and start the machine. Also, to check if the run was successful and to deactivate the machine again. These variables have been selected and are available in the system.

The main algorithm is split into two parts, the first part is to acquire data for training the model. In the second part of the algorithm the machine learning model, which has been trained before, is used to predict the result of each randomly generated time. If the outcome is predicted as true, the machine is started with the value. If the predicted outcome is false, a new time is generated till a true one was found. Within this algorithm a random time is generated.

Visualisation

```
def pickle_visualise():
    with open(pickle_name, 'rb') as pickle_file:
        table = pickle.load(pickle_file)

    times = []
    values = []
    for jj in range(len(table[0])):
        times.append(table[0][jj])
        values.append(table[1][jj])

    #  recombine array
    packed_data = np.vstack((times, values))
    #  Transform Array
    transformed_data = packed_data.T
    #  sort data by runtime
    sorted_data = sorted(transformed_data, key=lambda x: x[0])
    #  turn list result into numpy array
```

```
    data_array = np.asarray(sorted_data)
    #  extract x and y vectors for plotting
    x, y = data_array.T
    #  only keep successful runs
    #  create numpy array from y(y are True and False values from plotting)
    condition = np.array(y)
    #  transform to boolean
    condition = condition.astype(bool)
    #  extract only values with True condition in y Array from x Array to
rematch True values with their times
    #  return only successful runs
    pos_values = np.extract(condition, x)

    #  get average, min and max successful Delay Time
    ave_values = round(pos_values.mean(), 4)
    min_values = pos_values.min()
    max_values = pos_values.max()
    text = "Average: %s Min: %s Max: %s" % (ave_values, min_values,
max_values)
    print(text)

    #  plot Results and print min,max,average to plot
    plt.plot(x, y, 'bo', x, y, 'b', linewidth=2.5)
    plt.plot(min_values, 1, 'ro')
    plt.plot(max_values, 1, 'ro')
    plt.title('Test Results\n %s' % text)
    plt.grid(True, which='major')
    plt.ylim(-0.5, 1.5)
    plt.xlim(2, 5)
    plt.xlabel('Time')
    plt.ylabel('Result')
    plt.yticks([0, 0.5, 1])
    plt.axhline(0, color='grey')
    plt.axhline(1, color='grey')
    plt.show()
```

This function is used to analyse and visualise the results saved in the pickle database file. Firstly, the pickle is opened and its array containing all run times and results loaded. To allow visualisation using the Pyplot module the data must be arranged in a specific shape. The times and results are sorted by increasing numbers. If they would stay in their random order the plotting operation won't follow a straight line which would have a negative impact on the readability.

The runs are also mathematically analysed. Only successful runs with the true parameter are kept in a separate array. From this array the average successful runs, lowest successful and highest successful time are extracted and saved. These are displayed in the command line and will, additionally be highlighted in the plot window. The plotting operation is carried out and several visual modifications are made to enhance readability.

User Interface

```
master = Tk()
master.title('Machinery Start-up')
sel = IntVar()
# row 0
Label(master, text='This program tries to find the right start-up
time').grid(row=0, column=1)
# row 1
Label(master).grid(row=1)
# row 2
Label(master, text="Input number of desired training runs:").grid(row=2,
column=1)
# row 3
e1 = Entry(master)
e1.grid(row=3, column=1)
# row 4
Label(master).grid(row=4)
# row 5
Label(master, text="Input number of desired test runs:").grid(row=5, column=1)
# row 6
e2 = Entry(master)
e2.grid(row=6, column=1)
# row 7
Label(master).grid(row=7)
# row 8
Label(master, text="Run on Machine:").grid(row=8, column=1)
# row 9
R1 = Radiobutton(master, text="Machine 1", indicatoron=1, variable=sel,
value=1).grid(row=9, column=1)
# row 10
R2 = Radiobutton(master, text="Machine 3", indicatoron=1, variable=sel,
value=3).grid(row=10, column=1)
# row 11
Label(master).grid(row=11)
# row 12
Button(master, text='RUN', fg="white", bg="darkgreen", command=main_algo,
height=2, width=40).grid(row=12, column=1)
# row 13
Button(master, text='EMPTY', fg="white", bg="orange",
command=create_new_pickle, height=2, width=40).grid(row=13,
column=1)
# row 14
Button(master, text='VISUALISE', fg="white", bg="#2A4E6C",
command=pickle_visualise, height=2, width=40).grid(row=14,
column=1)
# row 15
Button(master, text='EXIT', fg="white", bg="red", command=master.destroy,
height=2, width=40).grid(row=15, column=1)

master.mainloop()
```

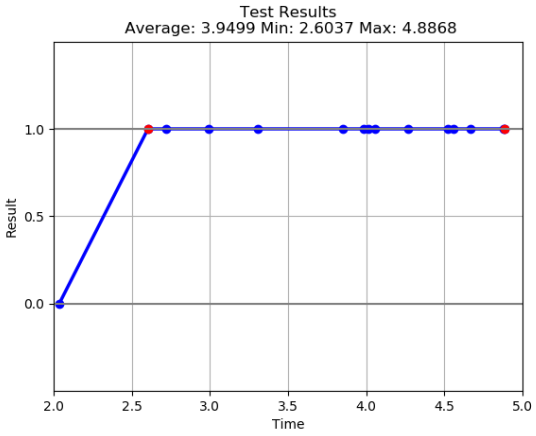
The Code for the user interface must be placed at the very end of the file. Firstly a “master” window is introduced and labelled. Similarly at the very end the master window is ended with “master.mainloop()”. The User Interface that is displayed in the window is organised in a grid

layout, allowing smooth and fast positioning of its components. Label fields allow for the displaying of text, entry fields allow the user to enter values for the program. The values for desired training and test runs must be entered by the user. Buttons are intruded for the rows eight to eleven. Buttons are used to call specific functions from the code. The “Run” will call the programs main algorithm to get the execution of the program started. Bellow the “Empty” button will delete the pickles database content. This function is useful when a new dataset is collected. Next up in row ten the “Visualise” button calls the visualisation function. Lastly the “Exit” Button will close the active window.

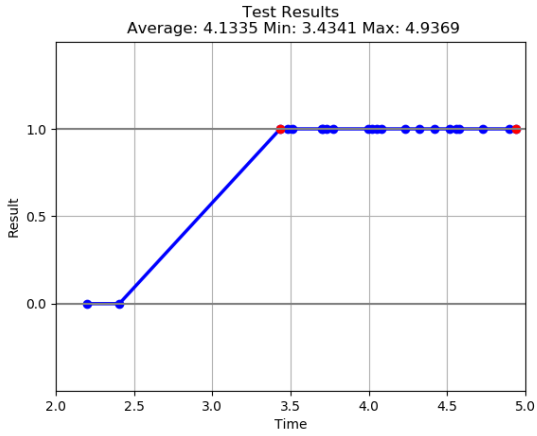
Results

Interpretation

The Program was executed for the machines MG1 and MG3 in the CDHAWs Industry 4.0 laboratory. The program ran flawlessly and delivered the expected results. In the created diagrams below, it is visible that there is a requirement for a waiting period on both machines. The minimum waiting time for machine one is 2.6037 seconds and 3.4314 seconds for machine three. This is visible in the plot attached below. Different to the expectations there is no maximum for a waiting period. From our test runs it is also visible that the average successful waiting period for machine one is 3.9499 seconds and for machine three 4.1335 seconds. These values are based on the completed runs.



Results Plot for Machine 1



Results Plot for Machine 3

Both machines show a maximum value close to five seconds. This indicates that there is not maximum waiting period. More runs will improve the accuracy and will give an improved result,

as machine learning is based on large datasets. This especially as a machine in an industrial environment needs to work every time. The minimum value found during the test-runs may not work every time, additional security is needed. It is advisable to use the average successful delay time.

Conclusion

To conclude, Industry 4.0 holds immense potential to change the world of production persistently. The opened opportunities to further raise the efficiency of production systems and the idea of fully autonomous manufacturing lines.

Lessons Learnt

During this project we learned a lot about the Python programming language. From the most basic functions to more advanced ones covering mathematics, visualisation and PLC connections.

Machine Learning Applications

Industrial Processes collect large datasets along their supply and value chain. With machine learning these data sets can be analysed effectively with minimal resources. This allows to further improve processes using this data.

Machine Learning can be applied not only to industrial processes but to everything that involves data. For example, it can be used in the farming industry to increase the output of agricultural production.

Fields of application are widely distributed along the industrial environment. With increasing automation in production productivity and quality can be improved and therefore reduce costs related to the production.

Increased research in machine learning will allow for more applications and improved results.

Appendix

```
import OpenOPC
import numpy as np
import cPickle as pickle
import time
from Tkinter import *
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
# Set up connection to OPC
opc = OpenOPC.client()
opc.connect('PhoenixContact.AX-Server.21')
time.sleep(2)


# Creating new pickle-file
def create_new_pickle():
    with open(pickle_name, 'wb') as pickle_file:
        pickle.dump([[], []], pickle_file)


# Writing results and time into pickle
def ins_pickle():
    global current_p_list

    while True:
        try:
            with open(pickle_name, 'rb') as pickle_file:
                current_p_list = pickle.load(pickle_file)

            for jj in range(len(times_current_run)):
                current_p_list[0].append(times_current_run[jj])
                current_p_list[1].append(results_current_run[jj])

            with open(pickle_name, 'wb') as pickle_file:
                pickle.dump(current_p_list, pickle_file)

            break

        except IOError:
            create_new_pickle()


# Test run
def test_run():
    global times_current_run
    global results_current_run
    times_current_run = []
    results_current_run = []

    times_current_run.append(t1)
    opc.write([machine_unlock, True])
    time.sleep(t1)
    opc.write([machine_on, True])
    time.sleep(7.5)
    result = opc[machine_check]
    print(result)
    results_current_run.append(result)
    opc.write([machine_off, True])
    time.sleep(7.5)

    ins_pickle()


# Machine Learning Algorithm
def scikit_eval():
```

```
    global clf
    global table
    table = []

    with open(pickle_name, 'rb') as pickle_file:
        table = pickle.load(pickle_file)
        x = table[0]
        y = table[1]
    # n_neighbors refers to the number of neighbors to be tested
    clf = KNeighborsClassifier(n_neighbors=2)
    # Generally execute Train_Test_Split by using 20 to 40% for testing
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=4)
    # Switch type from list to array
    x_train_a = np.asarray(x_train)
    x_test_a = np.asarray(x_test)
    y_train_a = np.asarray(y_train)
    y_test_a = np.asarray(y_test)
    # transpose the features matrix
    x_train_ar = x_train_a.reshape(-1, 1)
    x_test_ar = x_test_a.reshape(-1, 1)
    # Training the machine with the training data
    clf.fit(x_train_ar, y_train_a)
    # Testing the machine with the test data
    y_pred = clf.predict(x_test_ar)
    # Printing the result
    accuracy = metrics.accuracy_score(y_test_a, y_pred)
    print('The trained model has an accuracy of: ' + str(np.round(accuracy, 6)
* 100) + '%')


# Run Visualisation and display low, high value
def pickle_visualise():
    with open(pickle_name, 'rb') as pickle_file:
        table = pickle.load(pickle_file)

    times = []
    values = []
    for jj in range(len(table[0])):
        times.append(table[0][jj])
        values.append(table[1][jj])

    #  recombine array
    packed_data = np.vstack((times, values))
    #  Transform Array
    transformed_data = packed_data.T
    #  sort data by runtime
    sorted_data = sorted(transformed_data, key=lambda x: x[0])
    #  turn list result into numpy array
    data_array = np.asarray(sorted_data)
    #  extract x and y vectors for plotting
    x, y = data_array.T
    #  only keep successful runs
    #  create numpy array from y(y are True and False values from plotting)
    condition = np.array(y)
    #  transform to boolean
    condition = condition.astype(bool)
    #  extract only values with True condition in y Array from x Array to
```

```
rematch True values with their times
    #  return only successful runs
    pos_values = np.extract(condition, x)

    #  get average, min and max successful Delay Time
    ave_values = round(pos_values.mean(), 4)
    min_values = pos_values.min()
    max_values = pos_values.max()
    text = "Average: %s Min: %s Max: %s" % (ave_values, min_values,
max_values)
    print(text)

    #  plot Results and print min,max,average to plot
    plt.plot(x, y, 'bo', x, y, 'b', linewidth=2.5)
    plt.plot(min_values, 1, 'ro')
    plt.plot(max_values, 1, 'ro')
    plt.title('Test Results\n %s' % text)
    plt.grid(True, which='major')
    plt.ylim(-0.5, 1.5)
    plt.xlim(2, 5)
    plt.xlabel('Time')
    plt.ylabel('Result')
    plt.yticks([0, 0.5, 1])
    plt.axhline(0, color='grey')
    plt.axhline(1, color='grey')
    plt.show()


# Main algorithm
def main_algo():
    # machine selection
    global c
    global machine_unlock
    global machine_on
    global machine_check
    global machine_off
    global pickle_name
    global t1

    c = sel.get()
    if c == 1:
        machine_unlock = 'V2_MG1.Web_MachineUnlock_Button'
        machine_on = 'V2_MG1.Web_MachineOn_Button'
        machine_check = 'V2_MG1.O_xSt1_Motor_8_Anticlockwise'
        machine_off = 'V2_MG1.Web_MachineOff_Button'
        pickle_name = 'pickle_MG1.pkl'
        print(machine_unlock, machine_on, machine_check, machine_off)
    elif c == 3:
        machine_unlock = 'V2_MG3.Web_MachineUnlock_Button'
        machine_on = 'V2_MG3.Web_MachineOn_Button'
        machine_check = 'V2_MG3.OPC_xRobot_MotorOn_LED'
        machine_off = 'V2_MG3.Web_MachineOff_Button'
        pickle_name = 'pickle_MG3.pkl'
        print(machine_unlock, machine_on, machine_check, machine_off)
    else:
        print("ERROR")
    print("execution on Machine: ", c)
```

```
    # Training phase
    global e1
    global training_times
    training_times = int(e1.get())

    for z in range(training_times):
        t1 = round(np.random.uniform(2, 5), 4)
        test_run()

    # Testing phase
    global e2
    global test_times
    test_times = int(e2.get())

    scikit_eval()

    for v in range(test_times):

        while True:
            t1 = round(np.random.uniform(2, 5), 4)
            t1_r = np.array([t1])
            t1_r = t1_r.reshape(-1, 1)
            res_pred = clf.predict(t1_r)

            if res_pred == True:
                break

        test_run()


# Tkinter
master = Tk()
master.title('Machinery Start-up')
sel = IntVar()
# row 0
Label(master, text='This program tries to find the right start-up
time').grid(row=0, column=1)
# row 1
Label(master).grid(row=1)
# row 2
Label(master, text="Input number of desired training runs:").grid(row=2,
column=1)
# row 3
e1 = Entry(master)
e1.grid(row=3, column=1)
# row 4
Label(master).grid(row=4)
# row 5
Label(master, text="Input number of desired test runs:").grid(row=5, column=1)
# row 6
e2 = Entry(master)
e2.grid(row=6, column=1)
# row 7
Label(master).grid(row=7)
# row 8
Label(master, text="Run on Machine:").grid(row=8, column=1)
# row 9
R1 = Radiobutton(master, text="Machine 1", indicatoron=1, variable=sel,
```

```
value=1).grid(row=9, column=1)
# row 10
R2 = Radiobutton(master, text="Machine 3", indicatoron=1, variable=sel,
value=3).grid(row=10, column=1)
# row 11
Label(master).grid(row=11)
# row 12
Button(master, text='RUN', fg="white", bg="darkgreen", command=main_algo,
height=2, width=40).grid(row=12, column=1)
# row 13
Button(master, text='EMPTY', fg="white", bg="orange",
command=create_new_pickle, height=2, width=40).grid(row=13,
column=1)
# row 14
Button(master, text='VISUALISE', fg="white", bg="#2A4E6C",
command=pickle_visualise, height=2, width=40).grid(row=14,
column=1)
# row 15
Button(master, text='EXIT', fg="white", bg="red", command=master.destroy,
height=2, width=40).grid(row=15, column=1)

master.mainloop()
```