Wer hat gebohrt?

Release 21.01.2022

Tobia Ippolito, Syon Kadkade, Vadim Korzev

Jan 20, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

This library is programmed for distinguish drilling persons.

There is a need to have two specific programs to get and save the measurement of the driller. The programms called drilldriver and drillcapture.

If you want to take off with this project, check out the packages and see the Tutorial.

Our application have following features:

- Graphical-User-Interface
- Good usability
- Many possibilities (delete wrong drills, choose ml-algorithm and more…)
- Stunning graphics
- Easy to use

1.1 Possibilities of our future:

We have imagined a field of use, which can transfer to many other ideas.

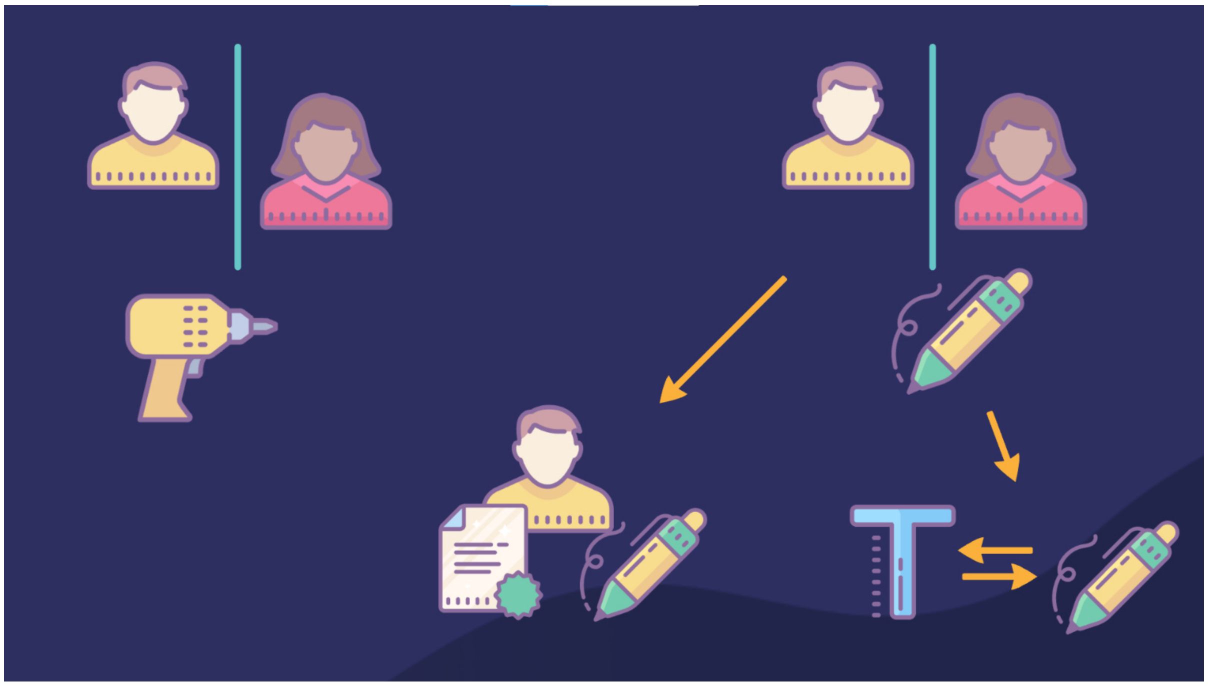
Hint: It is far fetched and it may contain traces of humour. But it should still be allowed to have dreams. With that philosophy, let us start.

We present the authentic-copy tool, the friend of every student. You have to do some complex math and there are not much time left? You only need one solution. The authentic-copy tool reads the content out of this solution and converts its in your own writing style! Nobody will think that you have written off! (For real: learn and do your stuff by yourself)

Watch the video: https://prezi.com/view/TvhPlx4Yl0dVGcCnpvs9/ on this presentation you can also see other things about our project!

But wheres the connection to our project, right? Our project and the authentic-copy-tool can distinguish 2 people with physical properties in another domain. We thought further: what is, if it is possible to represent the learned drill-style or writing-style on new content. That would be great. And for the copy-tool there is one further component to translate written text to digital text.

The follwoing image describes that, graphically.



1.2 Author's:

Tobia Ippolito -> https://github.com/xXAI-botXx

Syon Kadkade

Vadim Korzev

CHAPTER TWO

TUTORIAL

This library is programmed for distinguish drilling persons.

To start the application, you need a setup with following points:

- drilldriver and drillcapture programs
- wired drill
- wood for drilling
- at least 2 persons
- our library
- python3 installed
- all modules in the requirements.txt (see below)

```
###### Requirements without Version Specifiers ######
pandas
matplotlib
sklearn
ttkthemes
pyyaml
tsfresh
pillow
webbrowser

###### Requirements with Version Specifiers ######
numpy <= 1.20

###### Linux System need to install ######
# Tkinter
# sudo apt-get install python3-tk

# And Pillow Tk
# sudo apt-get install python3-pil.imagetk
```

If the setup is ready, you have to call the run-function in following module:

anoog.automation.graphical_user_interface

See the RUN.py for an exemplary call.

Hint: Dabei muss sichergestellt sein, dass kein weiterer Drilldriver läuft. Um das zu tun, muss einfach der Output in der Konsole/Terminal überprüft werden und es muss **MCCUDP found!** dastehen. Fall dort etwas steht wie **MCCUDP**

Acquiring… muss die Anwendung geschlossen werden und folgende Schritte befolgt werden:

1. Im Terminal **killall MCCUDP** eingeben
2. Erneut **killall MCCUDP** im Terminal eingeben
3. Anwendung mit **python3 RUN.py** starten
4. Anwendung schließen
5. Anwendung erneut starten und nun funktioniert alles wieder

CHAPTER

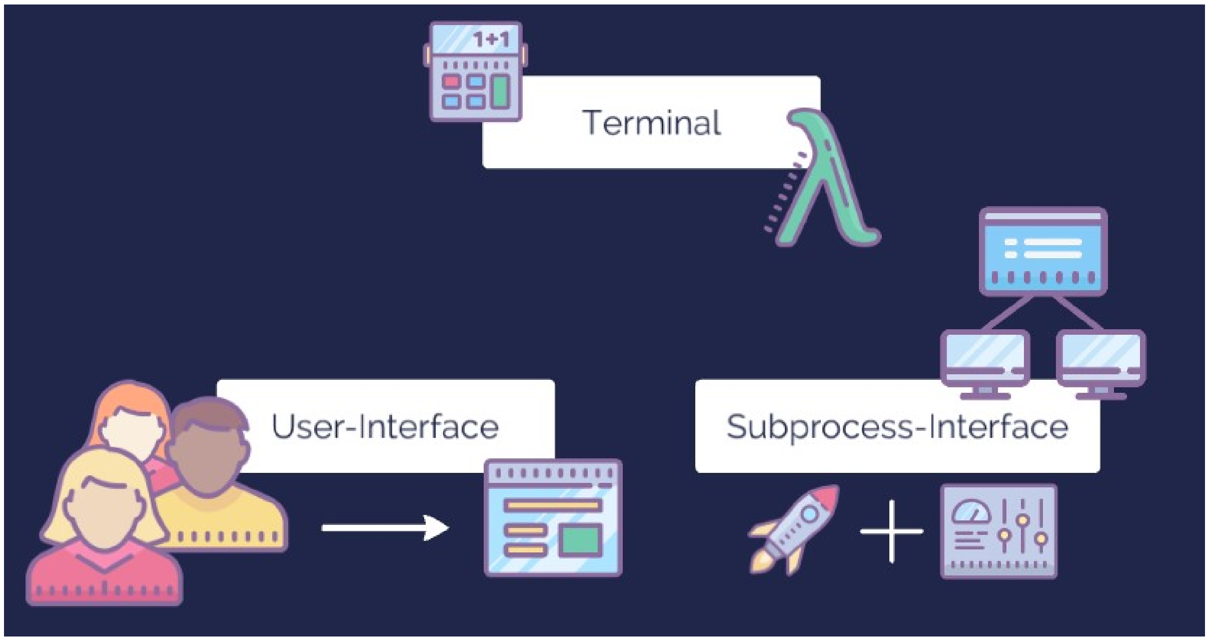
THREE

EXPLANATION

(In german)

3.1 Live-Application

Unser Programm besteht zur Laufzeit aus 3 großen Komponenten und wurde komplett in Python programmiert. Die GUI (Graphical-User-Interface) nimmt die Wünsche des Users entegen und gibt diese dem Terminal weiter. Das Terminal verarbeitet diese Wünsche, veranlasst das Laden der gesammelten Daten und auch dessen Training und Vorhersage. Das Subprocess-Interface hilft bei dem Starten und kommunizieren mit der Drilldriver und der Drillcapture Datei. Diese beiden Dateien sind für das sammeln und speichern der Bohrdaten von den Sensoren zuständig und wurden nicht von uns programmiert.

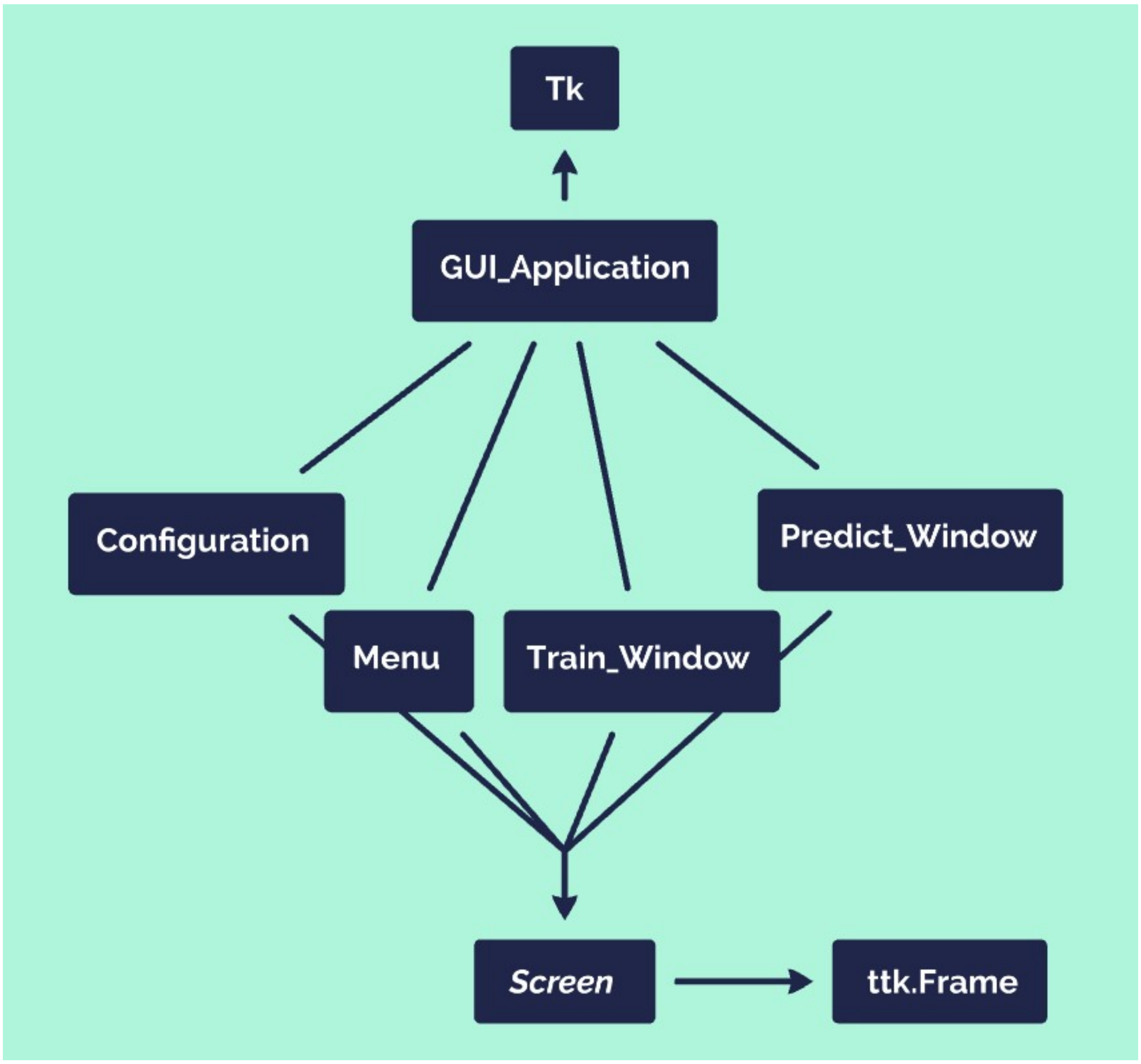


Und jetzt schauen wir uns diese 3 Komponenten etwas im Detail an. Es geht dabei nicht um die Implementierung.

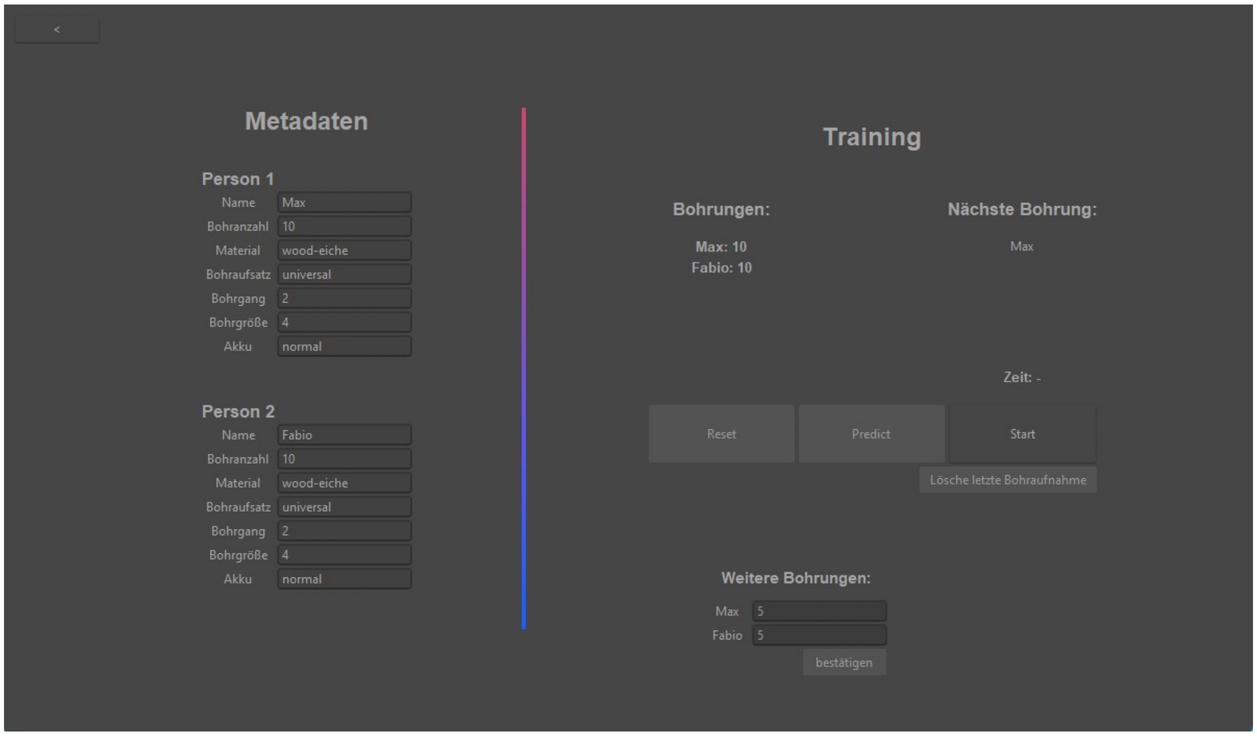
Die GUI

Die GUI wurde in der Python-GUI-Bibliothek Tkinter programmiert. Damit sie modern aussieht, wurden alle Komponent nicht normal implementiert, sondern von ttk. Hier wird das Design von dem Betriebssystem genommen. Als Entwickler hat man dadurch weniger Möglichkeiten, jedoch sehen die Widgets zeitgemäß aus. Außerdem wurde ttkthemes verwendet. Dieses Modul besitzt den Vorteil, dass man einen bestimmten Style wählen kann und alle Widgets sehen so aus. Wir haben uns für einen dunklen Style entschieden, was gut mit Farben kombiniert werden kann.

Bei Tkinter ist es so, dass es einen root gibt. Und jedes Widget muss einen Parent Widget haben (außer der root, dass ist der Startpunkt). Unsere GUI-Anwendung erbt von Tk und ist damit der root. Nun gibt es verschiedene Frames, wie das Startmenu oder der Bohrscreen zum Trainieren. Diese haben den root als Parent und erben selber von Screen, was ein ttk.Frame ist und ein paar wenige Dinge beinhaltet. Jeder Frame hat nun verschiede Children (Buttons, Labels, Textfields, . . .), welche dann dessen Inhalt sind.



Und so funktioniert unsere GUI auf abstrakter Ebene. Schau hierfür gerne in wer/src/anoog/automation/graphical_user_interface.py Bevor wir uns das Terminal ansehen, werden wir noch einen Ausschnitt von der GUI betrachten. Um sie einmal gesehen zu haben.



Metadaten

Person 1

Name	Max
Bohranzahl	10
Material	wood-eiche
Bohraufsatz	universal
Bohrgang	2
Bohrgröße	4
Akku	normal

Person 2

Name	Fabio
Bohranzahl	10
Material	wood-eiche
Bohraufsatz	universal
Bohrgang	2
Bohrgröße	4
Akku	normal

Training

Bohrungen:

Max: 10
Fabio: 10

Nächste Bohrung:

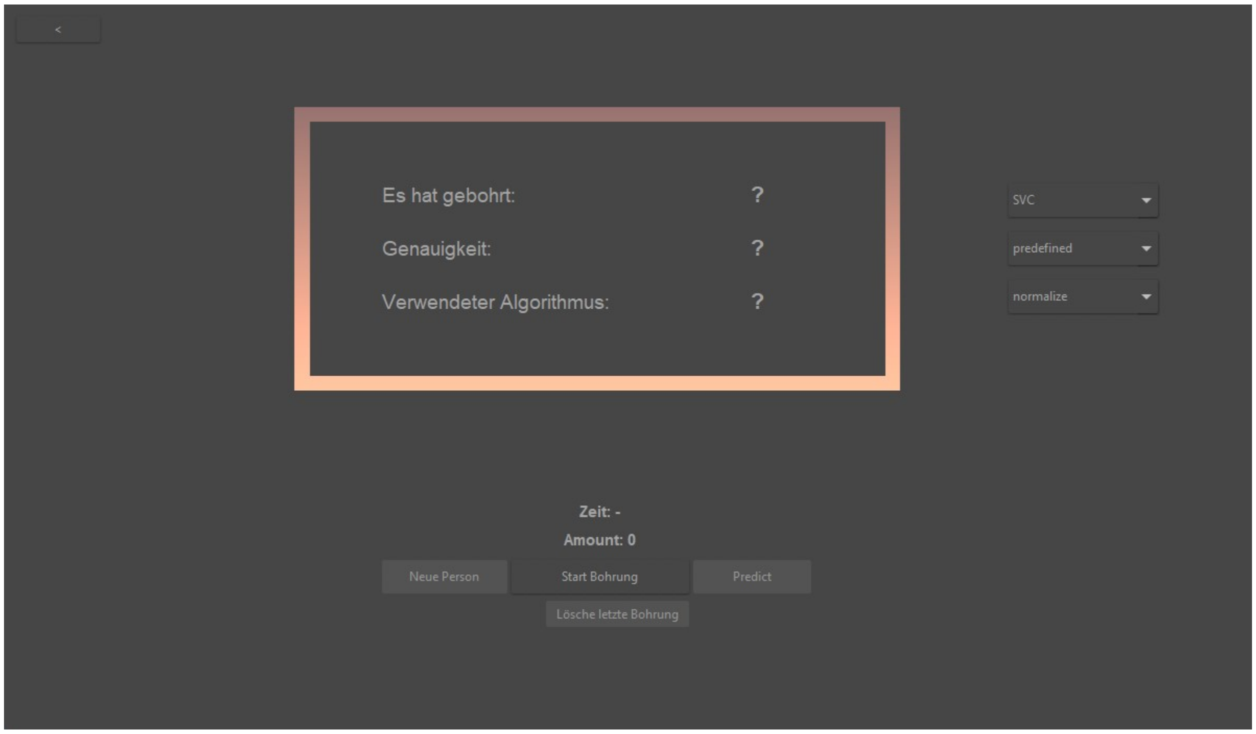
Max

Zeit: -

Weitere Bohrungen:

Max	5
Fabio	5

Links können sich die Benutzer mit den entsprechenden Merkmalen eintragen. Es ist auch schön zu sehen, wie spektakulär der Farbverlauf aus dem Dunkel hervortaucht. Rechts können die Bohrvorgänge / Sensoren gestartet und gestoppt werden. Es können neue Bohrungen hinzugefügt werden und falsche Bohraufnahmen gelöscht werden.



Es hat gebohrt: ?
Genauigkeit: ?
Verwendeter Algorithmus: ?

SVC
predefined
normalize

Zeit: -
Amount: 0

Bei diesem Screen können nun Testbohrungen durchgeführt werden und der Algorithmus versucht die Bohrung/en einem der Beiden zuzuordnen. Toll dabei ist, dass man hier herum experimentieren kann. Der Algorithmus ist nämlich wählbar. Die Hyperparameter kann man entweder voreingestellt (von uns) lassen oder sie herausfinden lassen (was aber risikoreich sein kann). Ob die Daten hierfür normalisiert werden sollen, kann man ebenfalls entscheiden.

Das Terminal

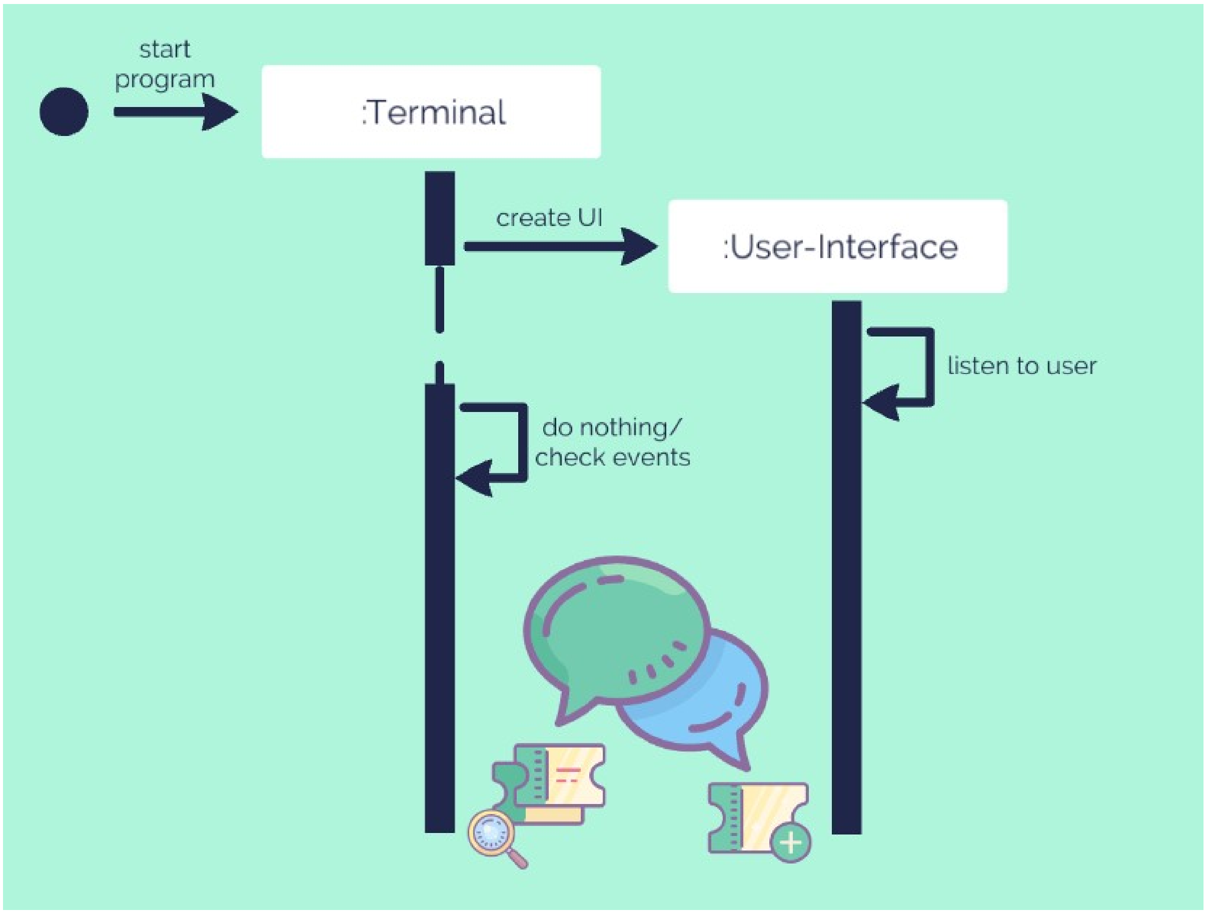
Das Terminal kümmert sich vor allem um die Logik. Beispielsweise um das Erstellen von den Ordnerstrukturen für das Speichern der Messdaten. Aber das Terminal lädt auch die gespeicherten Daten und lässt diese von einem Machine-Learning-Algorithmus trainioeren und predicten. Hierfür besitzt es ein AI-Model-Object. Dieses kümmert sich um genau um diese Dinge. Es gibt natürlich noch mehr zum Terminal zu sagen, aber alles weitere würde zu sehr ins Detail gehen. Hier gibt es für uns also ertsmal nichts weiter zu sehen und wir gehen zum Subprocess-Interface.

Das Subprocess-Interface

Um die Drilldriver und Drillcapture starten zu können verwenden wir das subprocess-Modul von Python. Hier verwenden wir vor allem das Objekt Popen. Mit dem kann man einen Terminal/Konsole mit einem Befehl starten. Und damit starten wir einmal die Drilldriver und dann noch die Drillcapture. Wichtiger Unterschied ist hier, auf welchem Betriebssystem man sich befindet.

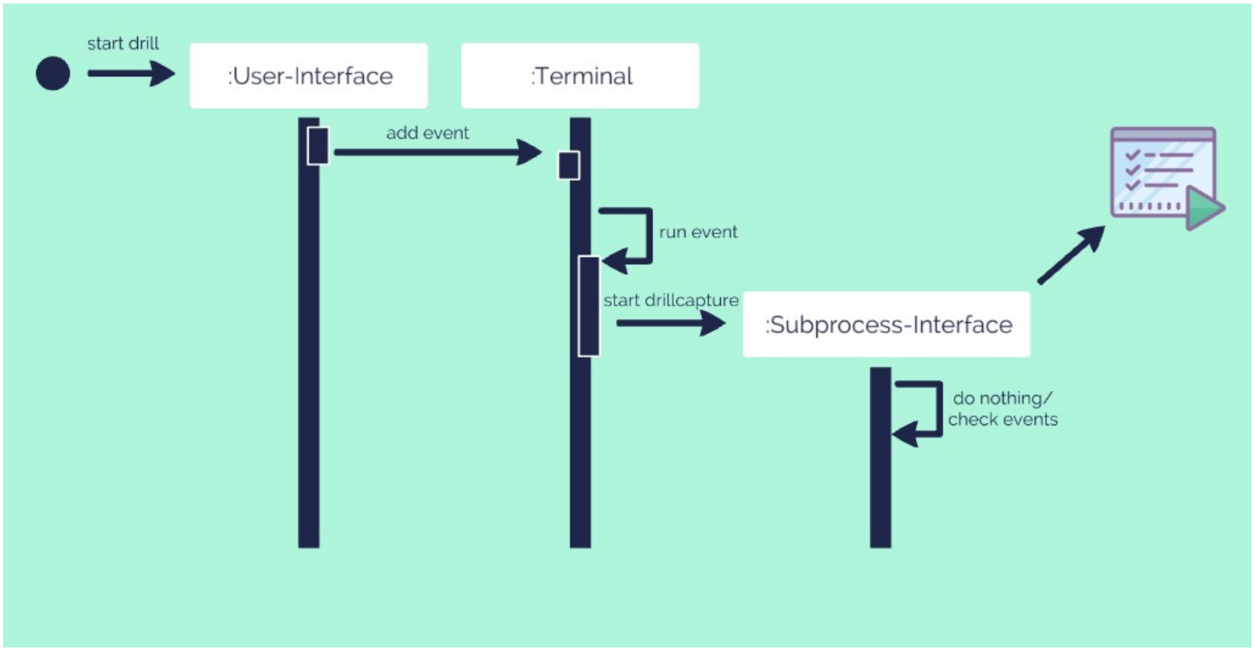
3.1.1 Laufzeit

Nicht unerwähnte sollte man den Code bei Laufzeit lassen. Wir haben 3 Komponenten, welche miteinander interagieren und gleichzeitg laufen müssen. Für uns heißt das, dass alle 3 Kompontenen in Threads laufen. Die Kommunikation währendessen läuft über ein Eventsystem.



Und wir schauen uns dieses Eventsystem etwas genauer an. Wenn man einem Objekt ein Event schicken möchte, ruft man die **add_event()**-Methode auf. Nun fügt das Objekt dieses Event in seiner Queue hinzu. Falls das Objekt das Event ausführen möchte, kann es es aus der Queue nehmen. Doch was ist ein Event eigentlich? In diesem Fall einfach nur ein

String bzw. ist repräsentiert dieser String einen Schlüssel in einer Hashmap. Und der Wert zu dem der Schlüssel führt ist eine Methode. EventHasmap[eventname] = event_function(). Vorteil hierbei ist, dass eine Hasmap sehr effizient ist. Das Objekt kann also seine Eventqueue prüfen (ob sie leer ist). Falls nicht kann das Objekt das Event als Schlüssel für dessen Hashmap verwenden und führt so die Zielmethode aus. Und das ist das Eventsystem. Hier eine Grafik, bei welcher man dieses Verhalten beobachten kann (zumindest teilweise):



Es soll ein Bohrvorgang gestartet werden. Der User drückt also einen Button, was den aufruf von add_event() beim Terminal nachsichzieht. Das Event was in der Queue vom Terminal gespeichert wird heißt beispielsweise ‘start_drill’. In der Hashmap gibt es zu diesem Schlüssel eine passende Funktion start_drill(), welche mit dem Event beim nächsten Mal Prüfen der Eventqueue aufgerufen wird. Und was dann folgt ist für das Eventsystem nicht relevant.

Im Folgenden kannst du dir die Implementierung ansehen. So kannman es vielleicht besser verstehen. Außerdem ist der Code Unabhängig von einem Sachverhalt und damit gut auf andere Probleme übertragbar.

Komponenten eines Eventsystem-Teilnehmers:

- Event-Queue (zum Sammeln der Events)

```
self.events = Queue()
```

- Event-Hashmap (zum Ausführen der Events)

```
self.EVENT = {'event1':self.func_for_event1, 'event2':self.func_for_event2}
```

- Eine Methode, um ein Event ausführen zu können

```
if not self.events.empty():
    event = self.events.get()
    # event = 'eventname' or event = 'eventname', *args
    if len(event) > 1:
    self.EVENT[event[0]](\*event[1])
    else:
    self.EVENT[event[0]]()
```

- Eine Methode, damit Andere ein Event hinzufügen können

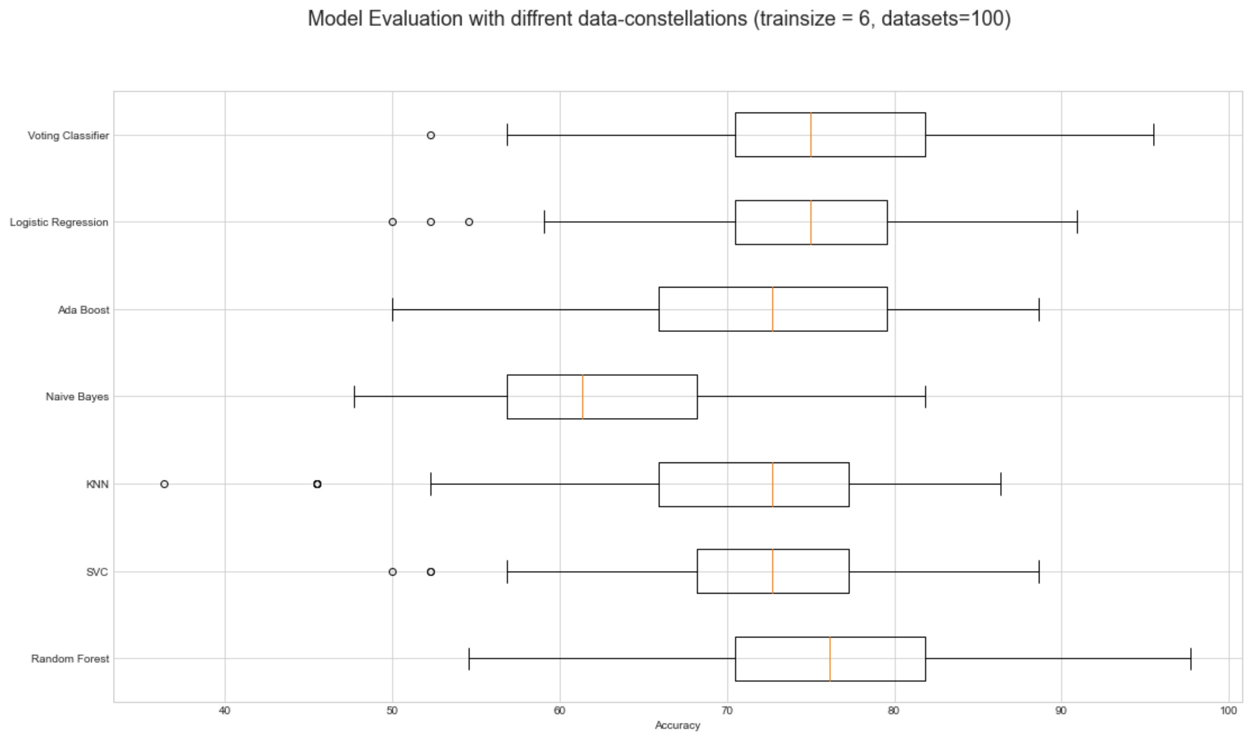
```
def add_event(self, event_name, \*additions):
      event = (event_name, \*additions)
      self.events.put(event)
```

3.2 AI-Model-Selection

Wir haben einige Machine-Learning Algorithmen unter realen bedingungen, sprich nur 6 Trainingsdaten, evaluiert.

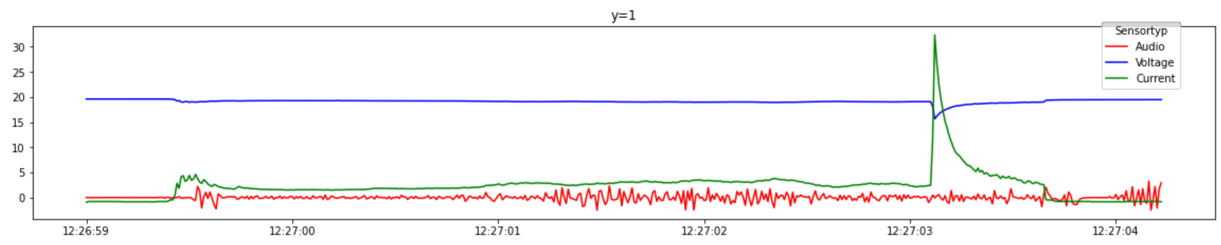
Die Accuracy der Algorithmen liegt sehr nah beinander und nur Naive Bayes ist eindeutig ungenauer. Und damit ist die Modelauswahl auch schon zuende, da man nun frei wählen kann.

Hint: Die Hyperparameter wurden seperat angepasst.

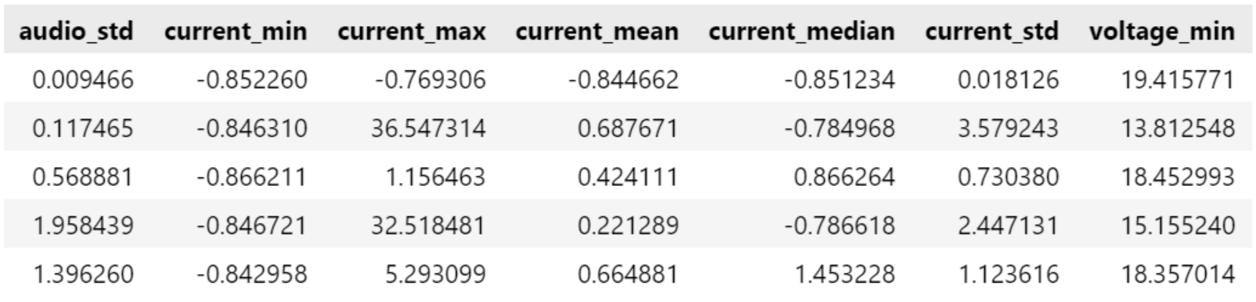


3.3 Timerow

Eine große Herausforderung war bei uns mit der Zeitreihe umzugehen. Welche etwa so aussieht:



Gelöst wurde dies mit der manuellen Feature-Extraction. Hierbei entnimmt man der Zeitreihe Eigenschaften, wie statische Werte. Die Classifier können nun mit diesen neuen Features ganz normal arbeiten.



audio_std	current_min	current_max	current_mean	current_median	current_std	voltage_min
0.009466	-0.852260	-0.769306	-0.844662	-0.851234	0.018126	19.415771
0.117465	-0.846310	36.547314	0.687671	-0.784968	3.579243	13.812548
0.568881	-0.866211	1.156463	0.424111	0.866264	0.730380	18.452993
1.958439	-0.846721	32.518481	0.221289	-0.786618	2.447131	15.155240
1.396260	-0.842958	5.293099	0.664881	1.453228	1.123616	18.357014

CHAPTER

FOUR

ANOOG

The library for distinguish drilling people.

4.1 anoog.automation

This Package contains all components for the live-application.

- GUI (& bg_booster)
- Controller/Terminal (& ai_model for holding logic)
- Subprocess-Interface

4.1.1 anoog.automation.ai_model

This module is used to hold the train/predict data and also to train/predict with a ML-Algorithm

Author: Tobia Ippolito

`class anoog.automation.ai_model.`**`AI_Model`**

Bases: `object`

This class used to:

- collect train- and predict-data
- train supervised classification ML-Algorithmn
- make prediction with the trained model

This class makes an protocoll of it tasks, to understand the progress. By init or resetting it clears this protocol.

`add_predict_dataset`(*person_id: int*, *person: str*, *data_path: str*)

Load last predict-drill-data in DataFrame. Using buffer with 30s timeout (if the data need some time to saving).

Parameters

- **`person_id`** (*`int`*) – ID of the Person who drill last time.
- **`person`** (*`str`*) – Name of the Person who drill last time.
- **`data_path`** (*`str`*) – Path to the data-directory. To know where collect the data.

`add_train_dataset`(*person_id: int*, *person: str*, *data_path: str*)

Load last train-drill-data in DataFrame. Using buffer with 30s timeout (if the data need some time to saving).

Parameters

- **person_id** (*int*) – ID of the Person who drill last time.
- **person** (*str*) – Name of the Person who drill last time.
- **data_path** (*str*) – Path to the data-directory. To know where collect the data.

`clean_log()`
Clears the protocol of the AI-Model.

Should be called once at the beinning of the program.

`draw_result`(*person1_name*, *person1_proba*, *person2_name*, *person2_proba*, *save_path*)
Plots the result of probability classification as bar-chart with 2 charts.

Parameters

- **str** (*- save_path as*) –
- **int** (*- person2_proba as*) –
- **str** –
- **int** –
- **str** –
- **person1_name** (*str*) – Name of the first person.
- **person1_proba** (*float or int*) – Probability of the first Person. Defines the height of one bar.
- **person2_name** (*str*) – Name of the second person.
- **person2_proba** (*float or int*) – Probability of the second Person. Defines the height of one bar.
- **save_path** (*str*) – Path, where the plot should be saved.

`predict()`
Predicts the predict-data with the trained model. If the train-data was normalize, the predict-data also do. Normalization is working with train-data, to have the same scale.

If there are more than one predict-data-entry. Every Entry going to predict and the probabilities are stacked (soft voting) and scaled to 100%.

Returns Returns the prediction probability

Return type tuple(float, float)

`remove_last_predict_dataset()`
Removes last predict-data-entry from DataFrame.

`remove_last_train_dataset()`
Removes last train-data-entry from DataFrame.

`remove_predict_dataset()`
Removes complete predict-data from DataFrame.

`remove_unused_columns`(*data*)
Tries to remove some irrelevant Features, which shouldn't exist anymore. It's no problem, if they don't exist.

Parameters **data** (*pd.DataFrame*) – Data, which will be trimmed

Returns Return new trimmed DataFrame

Return type pd.DataFrame

`reset()`
Sets the variables (like the train and predict-DataFrame) to initial values. (Clears also the protocol)

`train`(*algorithm=ALGORITHM.RANDOM_FOREST*, *auto_params=False*, *normalize=True*)
Train a model of choice. The Hyperparameters can choose as predefined or searched with GridSearchCV. And the data can be normalized if needed.

Note: If there are not enough data-entries, GridSearchCV don't be choosen. There have to be more than 5 entries.

Parameters

- **`algorithm`** (*`int, optional`*) – To know which ML-Algorithmn to choose
- **`auto_params`** (*`str, optional`*) – To know if Hyperparameter choose predefined or GridSearchCV
- **`normalize`** (*`str, optional`*) – To know if the data should be normalized

`write_log`(*message: str*)
Writes a String in the protocol-file.

This method should be used to add a entry to the protocoll.

Parameters **`message`** (*`str`*) – Message which will be saved in log.

`class anoog.automation.ai_model.`**`ALGORITHM`**(*value*)
Bases: `enum.Enum`

An enumeration.

`ADA_BOOST = 6`

`KNN = 3`

`LOGISTIC_REGRESSION = 5`

`NAIVE_BAYES = 4`

`RANDOM_FOREST = 1`

`SVC = 2`

`VOTING_CLASSIFIER = 7`

4.1.2 anoog.automation.bg_booster

This module contains a Tkinter-Widget for gradient-color.

Its runnable to test the Gradient-Color-Widget.

Author: Tobia Ippolito

`class anoog.automation.bg_booster.`**`Color_Gradient_Booster`**(*root*, *parent*, *mode='HORIZONTAL'*, *should_change_color=False*, *change_time=0.1*, *width=None*, *height=None*, *shiny_flow_effect=False*, *color_chain=None*, *gradient_size_tendency='neg'*)
Bases: `tkinter.Canvas`

Tkinter-Widget filled out with a gradient color.

The gradient-color can be static or dynamic (with a flow).

Parameters

- `root` (*tk.TK()*) – The basic widget of the gui-application.
- `parent` (*tk.Widget*) – The parent of this widget (which contains this widget).
- `mode` (*str, optional*) – Defines the direction of the gradient-color. ('HORIZONTAL', 'random', 'VERTICAL')
- `should_change_color` (*bool, optional*) – Defines whether or not the color changes (dynamic).
- `change_time` (*float, optional*) – Defines how fast the colors should be changed in dynamic mode. In seconds.
- `width` (*int, optional*) – The width of the widget (by most of the LayoutManager it's not relevant)
- `height` (*bool, optional*) – The height of the widget (by most of the LayoutManager it's not relevant)
- `height` – Activates/Deactivates a small variation of the color-values, should ended in a shiny-glimmer-effect.
- `color_chain` (*list of str, optional*) – Defines the used colors.
- `gradient_size_tendency` (*str, optional*) – Defines the direction of the flow (dynamic). ('neg' or 'pos' or 'random')

`calc_shiny_flow_effect()` → float

Calculates the variation of the color-value for the shiny-effect.

Returns Number of the variation of the normal color-point.

Return type float

`color_chain_change()`

Changes the current 2 colors using the color-chain.

With that, there can be many color used, but at one time can only 2 colors be activate.

`hex2rgb(`*str_hex*`)`

Calculate a hex color to an rgb color.

Parameters `str_hex` (*str*) – A color in hex-format with # or not at the beginning.

`run()`

Calls the update method and set itself to be called in some time (change_time defines that waiting time).

`set_fav_color()`

Picks a random color-chain from a selection.

Sets the new colors as color_chain inplace.

`set_off_screen()`

Unplace the Gradient-color-widget on his parent.

`set_on_screen()`

Place the Gradient-color-widget on his parent.

Will outfull the whole parent-widget.

`set_random_colors`(*n_colors=2*)
Calculates random colors.

Sets the new colors as color_chain inplace.

Parameters **`n_colors`** (*`int, optional`*) – Defines how many random colors should be calculated.

`start`()
Starts the flow-effect.

So that the gradient-color moves.

`stop`()
Stops the flow-effect.

So that the gradient-color not moves anymore.

`update`()
Creates and draws the gradient color.

`variate_color`()
Brings a little variation of the colors.

`variate_gradient_size`()
Variates the size of the 2 colors.

This creates a flow-effect.

`variate_gradient_size_speed`() → float
Variate the speed of the gradient-color change time.

4.1.3 anoog.automation.controller

This module is used to control the logical while running the application.

Author: Tobia Ippolito

`class` `anoog.automation.controller.`**`Terminal`**(*user_interface*, *data_path='data/testdata'*, *drillcapture_path='../BACKUP/drill-soft.exe'*, *drilldriver_path='../BACKUP/drill-soft.exe'*, *op=op.WINDOWS*)

Bases: *`anoog.automation.event.Eventsystem_Component`*

The Terminal class takes care of the logic of the live-application (who-drills). It creates the folders and files, controlls the subprocess (drillcapture and drilldriver), owns the data and AI-Model and counts the amounts of drilling.

This class sends Event-Messages to the GUI and the SubprocessInterface.

It runs with the run-Method and should run in a Thread. (Normally this is the startpoint of the program, but Tkinter has to run in the real program thread, so the GUI is the startpoint)

Parameters

- **`user_interface`** (*`GUI_App`*) – An interface to the user. For sending events (for example: show next drill-person)
- **`data-path`** (*`str, optional`*) – A path to the location where the data will be stored (there will be created a new folder with the current date)
- **`drillcapture_path`** (*`str, optional`*) – A path to the location where the Drillcapture program executable is stored.

- **drilldriver_path** (`str, optional`) – A path to the location where the Drilldriver program executable is stored.
- **op** (*op*, optional) – Defines on which operating system the program should run.

`add_amount`(*amount_person_1: str*, *amount_person_2: str*)
Increased Amount by 1. Prepares for new train-drill, if amounts over 0.

Parameters

- **amount_person_1** (`str`) – Includes the amount of drills left for the first person
- **amount_person_2** (`str`) – Includes the amount of drills left for the first person

`clean_output`()
Clear Output Files of Subprocess.

Should be called by start of the application.

`delete_last`()
Removes last train-data-entry in DataFrame and the real data (folder). Moreover the Amount is resetting by 1.

`delete_last_predict_drill`()
Removes last predict-data-entry in DataFrame and the real data (folder). Moreover the Predict-Amount is decreased by 1.

`exit`()
Stops the drilldriver and the drillcapture. Stable against None.

`from_predict_to_train`()
Sets the state, if user goes from predict-screen back to train-screen

`init_drill`(*person1: dict*, *person2: dict*)
Creates the Directories and yaml-Files for UNKNOWN and the two persons.

The yaml-Files contains basic information about the person who drills. This will be partly used as label, for the training of the supervised classification models.

Parameters

- **person1** (`dict`) – Includes informations about the first person. Have to store following keys: 'material', 'boreholeSize', 'gear', 'batteryLevel', 'drillType', 'operator', 'amount'
- **person2** (`dict`) – Includes informations about the second person. Have to store following keys: 'material', 'boreholeSize', 'gear', 'batteryLevel', 'drillType', 'operator', 'amount'

`init_predict`(*person1: dict*, *person2: dict*)
Creates the Directories and yaml-Files for UNKNOWN with dummy-informations.

Parameters

- **person1** (`dict`) – Includes informations about the first person. Have to store following keys: 'material', 'boreholeSize', 'gear', 'batteryLevel', 'drillType', 'operator', 'amount'
- **person2** (`dict`) – Includes informations about the second person. Have to store following keys: 'material', 'boreholeSize', 'gear', 'batteryLevel', 'drillType', 'operator', 'amount'

`init_train_drilling`(*person1: dict*, *person2: dict*)
Prepares the Application for a new train-drill. Choose a person for drilling. If both have amount left, pseudo-random will choose a person.

Parameters

- `person1` (*dict*) – Includes informations about the first person. Have to store following keys: 'material', 'boreholeSize', 'gear', 'batteryLevel', 'drillType', 'operator', 'amount'
- `person2` (*dict*) – Includes informations about the second person. Have to store following keys: 'material', 'boreholeSize', 'gear', 'batteryLevel', 'drillType', 'operator', 'amount'

`load_predict()`
Change the internal state to predict. Important for some methods.

`predict`(*model: str*, *mode: str*, *normalize: str*)
Train a given model with given parameters with the collected data. After that, the model will predict the predict-data (every entry). Get Soft Voting and update GUI with Results.

Parameters

- **`model`** (*str*) – Sets the model which will be used for training and prediction. Following Values exists: 'RandomForest', 'SVC', 'Naive Bayes', 'KNN', 'Ada Boost', 'Logistic Regression'
- **`mode`** (*str*) – Defines whether uses predefined hyperparameter for the model or use Grid-Search ('auto' or not).
- **`normalize`** (*str*) – Decided if the data should be normalized or not ('normalize' or not).

`reset_predict()`
Sets all variables with predict-context to the init values. Not includes the train-data (for purpose).

`reset_train()`
Sets all variables with train-context to the init values. Not includes the predict-data (for purpose).

`run()`
First cleans the Output-Files of the drillcapture file.

Then check it event-queue for new events and process it, if there is one. There is a short waiting buffer.

`start()`
Starts a drill (train or predict) with Drillcapture in single-mode.

`start_drill_driver()`
Starts the Drilldriver-Program.

Should be started one time by the start of the live-application.

`stop()`
Stops a drill (train or predict). Will check by train, if there is amount left and if not, it activates the predict area. Prepares next drill, if there is amount left.

`stop_drill_driver()`
Stops the Drilldriver-Program by sending a exit-event.

`class anoog.automation.controller.`**`process_state`**(*value*)
Bases: `enum.Enum`

An enumeration.

`BEFORE_TRAIN = 2`

`NOT_STARTED = 1`

`PREDICT = 5`

`TRAIN = 3`

`TRAIN_END = 4`

4.1.4 anoog.automation.event

This module provides content for members of the eventsystem.

The eventsystem is the possibality to commincate over Threads and lets run a method in another Thread.

Author: Tobia Ippolito

`class anoog.automation.event.`**`Eventsystem_Component`**

Bases: `object`

A member in the eventsystem should inherite from this class.

Funtionality: Other objectes/methods can call add_event() to add a new Event. Now the spicific Eventmember can process this event in his Thread by calling run_event. The given event in the queue is a key in the EVENT HashMap. The value should be a method, which will be called in run_event.

The Event should be a String (the eventname/key in EVENT) and can add multi-params in a tuple. For one param you should call as follow: add_Event(eventname, (param1,))

Notice following points: - The variable EVENT (HashMap/dictionary) should be implemented in the specific Eventmember class. - Don't forget to call: EventSystem_Component.__init__(self) - call run_event (maybe in ja while-loop or in a observer method)

`add_event`(*event_name*, **additions*)

Add a new event for this specific eventmember. Parameters can added as normal Parameters or in a tuple.

The event_name should be a key in EVENT-HashMap.

Parameters

- **`event_name`** (*`str`*) – Key in the EVENT HashMap
- **`additions`** (*`dict, optional`*) – Additional arguments for the event function.

`run_event`()

Checks the event-queue and runs a event, if there is one.

The eventname used as key in the EVENT HashMap. The value should be a callable and this function/method is called. If there were params (in a tuple) then they will handed over.

4.1.5 anoog.automation.graphical_user_interface

This module is used to implement a graphical user interface for the application. It contains all Widget/Screen-Classes for GUI. Only the Gradient-Color-Widget has it own class :module:~anoog.automation.bg_booster.py -> and implements the start-method of the application

Author: Tobia Ippolito

`class anoog.automation.graphical_user_interface.`**`Credits_Window`**(*root*, ***kwargs*)

Bases: *`anoog.automation.graphical_user_interface.Screen`*

Contains the logic and all widgets from credits-screen.

This Screen used to show all contributers of this project.

Parameters

- **`root`** – Root-Widget. Needed for communicate with :class:~anoog.automation.controller.Terminal.
- **`root`** – tk.Tk
- **`kwargs`** (*`dict`*) – Key, Value arguments for the Widget (relatively unrelevant)

add_padding()
Defines all paddings of the widgets.

create_widgets()
Method to create all important Widgets of the Credit-Screen.

event_back()
Load the start-menu.

hide()
Hides the Credits-Screen.

For that the screen will be unpacked and the animation will be stopped.

show()
Shows the Credits-Screen.

For that the screen will be packed and the animation will be started.

update_show()
Updates the position of the labels.

With that method called many times, ended in a flow effect.

weighting()
Defines all weightings of the widgets.

The weighting defines, how intensive the grid-entry fit, if the size of the screen is changing.

class anoog.automation.graphical_user_interface.**GUI_App**(***kwargs*)
Bases: `tkinter.Tk`, *anoog.automation.event.Eventsystem_Component*

This is the root-Widget of the GUI.

All other Windows will call and managed drom this root-point.

Parameters **kwargs** (*dict*) – Key, Value arguments for the Widget (relatively unrelevant)

check_events()
Checks the event-queue and calls to do so after 50ms again.

close()
Close the GUI and send an event to terminal to close it too.

event_resize_bg(*event*)
An event to fit the background-image to the screen-size.

Will be called every time, if the screen-size is changing. Only change image-size, if the size has changed. (The method also will be called, if the user moves the application.)

Parameters **event** – The event from Tkinter for resizing.

event_space(*event*)
Event handling for pressing Spacebar. Used for starting/stopping a new drill-process.

Parameters **event** – The event from Tkinter for Key-Pressed.

load_screen_credits(*from_widget*)
To load the Credit-Screen.

Parameters **from_widget** (*:class:~anoog.automation.graphical_user_interface.Screen*) – The current screen, which will be hided.

load_screen_main_menu(*from_widget*)
To load the Startmenu-Screen.

Parameters **from_widget** (*:class:~anoog.automation.graphical_user_interface.Screen*) – The current screen, which will be hided.

load_screen_predict(*from_widget*)
To load the Predict-Screen.

Parameters **from_widget** (*:class:~anoog.automation.graphical_user_interface.Screen*) – The current screen, which will be hided.

load_screen_train(*from_widget*)
To load the Train-Screen.

Parameters **from_widget** (*:class:~anoog.automation.graphical_user_interface.Screen*) – The current screen, which will be hided.

load_theme(*name*)
To load a Theme. Coloring and looking of the Widgets.

See the THEMES list for some Themes examples.

Parameters **name** (*str*) – Name of the theme.

reset_predict()
Resets the Predict-Screen. Uses internaly the :meth:~anoog.automation.graphical_user_interface.Predict_Window.event_rese
method.

Clears all drill-data, setted settings and showed results.

run(*data_path*, *drillcapture_path*, *drilldriver_path*, *op*, *path_to_project*)
The Startmethod of the Application.

Starts the Terminal and the GUI.

Parameters

- **data-path** (*str, optional*) – A path to the location where the data will be stored (there will be created a new folder with the current date)
- **drillcapture_path** (*str, optional*) – A path to the location where the Drillcapture program executable is stored.
- **drilldriver_path** (*str, optional*) – A path to the location where the Drilldriver program executable is stored.
- **op** (*op*, optional) – Defines on which operating system the program should run.
- **path_to_project** (*str, optional*) – A path to the location to the Project folder.

set_bg(*img*)
Can be used to set a background-image.

The size will be fitted on the screen. It uses a Label, which every screen owned.

(Currently not used)

Parameters **img** (*str*) – A Path to an image to load in Background.

set_min()
Sets the minimum height and width off a screen. The minimum is calculated, that all widgets have enough size and are completly showed.

This calculation is automaticly.

set_theme(*name: str*)
Set the Theme of the GUI. Standart is a dark-theme ('equilux').

This method will called by load_theme()-method.

Parameters `name` (*str*) – Name of the theme.

`start_button`(*state*)
Method to change the state of the start-button in the current screen. If disabled, the button isn't pressable anymore.

Only in predict-Screen or in Train-Screen available.

Parameters `state` (*str*) – State in which the start-button should be switch: 'disabled' or 'enabled' available.

`stop_time`()
Event handling for pressing Spacebar. Used for starting/stopping a drill-process.

class `anoog.automation.graphical_user_interface.Menu`(*root*, ***kwargs*)
Bases: `anoog.automation.graphical_user_interface.Screen`

Contains all widgets from start-screen. 3 Buttons (start, informationen, credits), title-label, Gradientcolor-stripes.

Parameters

- `root` – Root-Widget. Needed for communicate with :class:~anoog.automation.controller.Terminal.
- `root` – tk.Tk
- `kwargs` (*dict*) – Key, Value arguments for the Widget (relatively unrelevant)

`add_padding`()
Defines all paddings of the widgets.

`create_widgets`()
Method to create all important Widgets of the Start-Menu.

`event_button_credits`()
Loads the credits-screen.

Should be called, if the Credits-Button pressed.

`event_button_info`()
Opens the README-File of our project.

Should be called, if the Informationen-Button pressed.

`event_button_start`()
Loads the train-screen.

Should be called, if the Start-Button pressed.

`event_resize_label`(*event*)
Change the label-font size of the title, if the screen-size has changed.

With that method, the title is in a right size.

Parameters `event` – The event from Tkinter for resizing.

`hide`()
Hides the Start-Menu screen.

For that the screen will be unpacked and the gradient-color stripes will be stopped.

`show`()
Shows the Start-Menu screen.

For that the screen will be packed and the gradient-color stripes will be started.

weighting()

Defines all weightings of the widgets.

The weighting defines, how intensive the grid-entry fit, if the size of the screen is changing.

class anoog.automation.graphical_user_interface.**Predict_Window**(*root*, ***kwargs*)

Bases: *anoog.automation.graphical_user_interface.Screen*

Contains the logic and all widgets from predict-screen.

Included the show of results, add-drill-amount area and the ml-model-selection area.

This Screen used to collect predict-data, predict these data and show the results of them.

Parameters

- **root** – Root-Widget. Needed for communicate with :class:~anoog.automation.controller.Terminal.
- **root** – tk.Tk
- **kwargs** (*dict*) – Key, Value arguments for the Widget (relatively unrelevant)

add_padding()

Defines all paddings of the widgets.

change_run_time()

Logic of timer.

Updates runned time and calls his self in 0.1s.

create_widgets()

Method to create all important Widgets of the Predict-Screen.

delete_last_btn_change(*state*)

Deletes the last predict-data-entry.

Sends an Event to the Terminal.

Parameters **state** (*str*) – State in which the delete-last-button should be changed ('disabled' or 'enabled' available)

draw_result(*img_path*)

Draws a plot of the result.

(Not in use)

Parameters **img_path** (*str*) – Path to the image to load.

drill_ends()

Changes the text of the start-button to Start Bohrung and stops the timer.

drill_starts()

Changes the text of the start-button to Stopp Bohrung and starts the timer.

event_back()

Loads the Train-Screen.

Event called by clicking the back-button.

event_delete_last_button()

Deletes the last predict-drill-data-entry.

Sends an Event to the Terminal.

event_predict_button()
Trains the selected ML-Model with the train-data, predict the new data with the trained model and show the results of that.

Sends an Event to the Terminal.

event_reset_button()
Prepares for a new person to predict.

Delets the old predict-data-entries and delets the old results.

Sends an Event to the Terminal.

event_start_button()
Starts or stops a drill.

The method takes care of the state of the buttons. (While drilling the user should not be able to leave the screen or delete the last drill)

hide()
Hides the Predict-Screen.

For that the screen will be unpacked and the gradient-color will be stopped.

reset()
Sets all widgets and contents of the initilized value.

set_amount(*amount*)
Sets the amount to the given value.

Parameters **amount** (`int or str`) – Amount of drills

show()
Shows the Predict-Screen.

For that the screen will be packed and the gradient-color will be started.

show_result(*who*, *how*, *what*)
Shows the result of the prediction.

Parameters

- **who** (`str`) – Who is the prediction of the 2 persons.
- **who** – How sure is the model (percentage).
- **who** – Which algorithm used for this prediction.

start_button_change(*state*)
Changes the state of the start-button.

Parameters **state** (`str`) – State in which the start-button should be changed ('disabled' or 'enabled' available)

stop_time()
Stops the timer.

weighting()
Defines all weightings of the widgets.

The weighting defines, how intensive the grid-entry fit, if the size of the screen is changing.

class `anoog.automation.graphical_user_interface.`**Screen**(*root*, ***kwargs*)
Bases: `tkinter.ttk.Frame`, `abc.ABC`

Base Class for all Screens.

Implements the functionality for setting a background-image. For that a label is initialized. And methods to set and resize an image.

Parameters

- **`root`** – Root-Widget. Needed for communicate with :class:~anoog.automation.controller.Terminal.
- **`root`** – tk.Tk
- **`kwargs`** (*`dict`*) – Key, Value arguments for the Widget (relatively unrelevant)

`resize_bg`(*img*)

Method to reset the image of the label.

Parameters **`img`** (*`PIL.ImageTk`*) – New Backgroundimage

`set_bg`(*img*)

Method to reset the image of the label.

Parameters **`img`** (*`PIL.ImageTk`*) – New Backgroundimage

`class anoog.automation.graphical_user_interface.`**`Train_Window`**(*root*, ***kwargs*)

Bases: *`anoog.automation.graphical_user_interface.Screen`*

Contains the logic and all widgets from train-screen.

Included the Meta-Data-Section, the gradient-color hyphen and the train-data-section.

This Screen used to collect train-data for the prediction.

Parameters

- **`root`** – Root-Widget. Needed for communicate with :class:~anoog.automation.controller.Terminal.
- **`root`** – tk.Tk
- **`kwargs`** (*`dict`*) – Key, Value arguments for the Widget (relatively unrelevant)

`add_padding`()

Defines all paddings of the widgets.

`change_init_state`(*state='disabled'*)

Changes the availability of the widgets of the meta-data area.

Parameters **`state`** (*`str`*) – State in which the widgets should be changed ('disabled' or 'enabled' available)

`change_run_time`()

Logic of timer.

Updates runned time and calls his self in 0.1s.

`change_train_person`(*person*)

Shows the persons-name who should drill at next.

Parameters **`person`** (*`str`*) – The person who should drill at next.

`create_widgets`()

Method to create all important Widgets of the Train-Screen.

`delete_last_btn_change`(*state*)

Disabled or enabled the button to delete the last drill.

Parameters **`state`** (*`str`*) – Sets the state of the delete-last-button ('disabled' or 'enabled' available)

`drill_amount_change`(*amount_person_1*, *amount_person_2*)
Updates the amount-label.

Parameters

- `amount_person_1` (*str or int*) – Drill-Amount of Person 1 which is left.
- `amount_person_2` (*str or int*) – Drill-Amount of Person 2 which is left.

`drill_ends`(*amount_person_1: int*, *amount_person_2: int*)
Changes the start-button text to Start and update the drill-amount of both persons.

Parameters

- `amount_person_1` (*int*) – Drill-Amount of Person 1 which is left
- `amount_person_2` (*int*) – Drill-Amount of Person 2 which is left

`drill_starts`()
Changes the start-button text to Stopp and starts the timer.

`event_back`()
Loads the Startmenu and resets the train-screen and the predict-screen.

Called by clicking the back-button.

`event_confirm_add`()
Add new data-amount.

`event_confirm_init_change`()
Checks if the meta-data are valid and if yes, shows the person who should drill at next.

`event_delete_last_button`()
Communicates with the Terminal to delete the last data-entry.

`event_predict_button`()
Loads the predict-screen.

Only apears, if there are no more data-drills left.

`event_reset_button`()
Sets the screen and all his content to a initialized state.

`event_start_button`()
Starts or stops a drill.

The method takes care of the state of the buttons. (While drilling the user should not be able to leave the screen or delete the last drill)

`hide`()
Hides the Train-Screen.

For that the screen will be unpacked and the gradient-color stripes will be stopped.

`init_changes`(*var*, *indx*, *mode*)
Creates the confitm-button by the meta-data area.

Called if detected changes.

Params automatically sets by Tkinter and are not relevant and not used.

`show`()
Shows the Train-Screen.

For that the screen will be packed and the gradient-color will be started.

start_button_change(*state*)
Disabled or enabled the button to start a drill.

Parameters **state** (*str*) – Sets the state of the start-drill-button ('disabled' or 'enabled' available)

stop_time()
Stops the timer.

weighting()
Defines all weightings of the widgets.

The weighting defines, how intensive the grid-entry fit, if the size of the screen is changing.

anoog.automation.graphical_user_interface.**run**(*data_path='src/DrillDummy/testdata'*, *drillcapture_path='src/DrillDummy/drillcapture.exe'*, *drilldriver_path='src/DrillDummy/drilldriver.exe'*, *op=op.WINDOWS*, *path_to_project='./'*)
Creates the GUI and starts the whole Application.

Parameters

- **data-path** (*str, optional*) – A path to the location where the data will be stored (there will be created a new folder with the current date)
- **drillcapture_path** (*str, optional*) – A path to the location where the Drillcapture program executable is stored.
- **drilldriver_path** (*str, optional*) – A path to the location where the Drilldriver program executable is stored.
- **op** (*op*, optional) – Defines on which operating system the program should run.
- **path_to_project** (*str, optional*) – A path to the location to the Project folder.

4.1.6 anoog.automation.py_exe_interface

This module is used to start and control the drillcapture and drilldriver programs.

Author: Tobia Ippolito

class anoog.automation.py_exe_interface.**Drillcapture_Interface**(*absolut_path_to_exe: str*, *path_to_output*, *terminal*, *operating_system=op.WINDOWS*, *name=None*)
Bases: *anoog.automation.py_exe_interface.Process_Interface*

Class to start a drillcapture and communicate with it.

Parameters

- **absolut_path_to_exe** (*str*) – Describes the path to the location of the executable file, which will be run in Popen.
- **terminal** (*Terminal*) – Control object, to send event messages back.
- **operating_system** (*op*, optional) – Defines the current operating system. Impoartant for internal decisions (how to run commands).
- **name** (*str, optional*) – Name of the person, who want to start a drill.

`run`(*mode='bulk'*)

Starts the program and start listing to incoming events.

Overides `run()` for a new argument.

Parameters **mode** (`str, optional`) – Defines the mode, in which the drillcapture-program will be started.

`start_program`(*mode='bulk'*)

Specific stratetgy, which will be called in `run()`.

Starts the Drilldriver.

Parameters **mode** (`str, optional`) – Defines the mode, in which the drillcapture-program will be started.

`class anoog.automation.py_exe_interface.`**`Drilldriver_Interface`**(*absolut_path_to_exe: str*, *terminal*, *operating_system=op.WINDOWS*)

Bases: `anoog.automation.py_exe_interface.Process_Interface`

Class to start a drilldriver and communicate with it.

Parameters

- **absolut_path_to_exe** (`str`) – Describes the path to the location of the executable file, which will be run in Popen.
- **terminal** (`Terminal`) – Control object, to send event messages back.
- **operating_system** (`op`, optional) – Defines the current operating system. Impoartant for internal decisions (how to run commands).

`start_program`()

Specific stratetgy, which will be called in `run()`.

Starts the Drilldriver.

`class anoog.automation.py_exe_interface.`**`Process_Interface`**(*absolut_path_to_exe: str*, *terminal*, *operating_system=op.WINDOWS*)

Bases: `abc.ABC`, `anoog.automation.event.Eventsystem_Component`

Basic class for starting and controlling a process. Uses the strategy-pattern. What means, that the specific start-method can be implemented by every startegy.

Parameters

- **absolut_path_to_exe** (`str`) – Describes the path to the location of the executable file, which will be run in Popen.
- **terminal** (`Terminal`) – Control object, to send event messages back.
- **operating_system** (`op`, optional) – Defines the current operating system. Impoartant for internal decisions (how to run commands).

`check_input`()

Checks the stdin.txt, if there is a message for the interface/program. (Currently not used)

`event_listener`()

Checks the events, what the `Terminal` communicate which this interface about.

`is_alive`() → bool

Checks if the program is already running. Don't works at all.

Returns If the program works

Return type bool

`process_status()`

Should give a feedback of the status of the program.

Returns Status of Program (*running*, *paused*, *start_pending*, *pause_pending*, *continue_pending*, *stop_pending*, *stopped* or *not_existing*)

Return type str

`read()`

Reads the incomming Stream in Output Stream. Saves it in a File.

Returns The content in the Outputstream

Return type str

`read_err()`

Reads the incomming Stream in Error Stream. Saves it in a File.

Returns The content in the Errorstream

Return type str

`run()`

Starts the program (with specified strategy) and listen to incomming events.

`save_in_file`(*txt: str*, *file='log.txt'*)

Saves a given message in a log file (doeasn't ovveride something)

Parameters

- **txt** (*str, optional*) – Message which will be write in the file.
- **txt** – Name of the log file.

`abstract classmethod start_program()`

Method to implement the specific strategy.

`stop`(*send_event=False*)

Stops the started program.

Parameters **send_event** (*bool, optional*) – If a event should be sended as reaction.

`write`(*reaction: str*)

Writes something in the inputstream of the started program. With that, the *Terminal* can communicate with the program. Started not by *Terminal*, instead of indirect start with a event.

Parameters **reaction** (*str*) – The Message, which the programm will gets.

`class anoog.automation.py_exe_interface.op`(*value*)

Bases: `enum.Enum`

An enumeration.

`LINUX = 2`

`WINDOWS = 1`

4.2 anoog.io

This Package has content about loading drill-data.

4.2.1 anoog.io.csv_io

This module used to load the drill data from csv.

Contains functions to load drill-data simply and without many features, created from drillcapture.

Author for `read_csv()`: Syon Kadkade Author for `load_single_data()`: Tobia Ippolito Author for `read_csv()` and the other functions: Stefan Glaser

class `anoog.io.csv_io.`**`loadData_mode`**(*value*)

Bases: `enum.Enum`

An enumeration.

`DASK = 2`

`NONE = 1`

`anoog.io.csv_io.`**`load_single_data`**(*person*, *data_path: str*) → pandas.core.frame.DataFrame

Loads a single drill-data created from drillcapture.

Uses the last-drill.

Parameters

- **`person`** (*`str`*) – The person, who drilled at last.
- **`datasetPath`** (*`str`*) – The path to the dataset, to load it.

Returns The measurement of one drill.

Return type tuple of pd.DataFrame

`anoog.io.csv_io.`**`load_tsfresh`**(*datasetPath*, *seriesIDs*, *csvName='capture.csv'*, *metaName='meta.yaml'*)

Loads a complete drill-data created from drillcapture.

Parameters

- **`datasetPath`** (*`str`*) – The path to the dataset, to load it.
- **`seriesIDs`** (*`list of str`*) – The operators/folder names which should be loaded.
- **`csvName`** (*`str, optional`*) – The name of the measurement file.
- **`metaName`** (*`str, optional`*) – The name of the measurement metadata file.

Returns The measurement and the metadata of all drills.

Return type tuple of pd.DataFrame

`anoog.io.csv_io.`**`read_csv`**(*csvFile*, *mode=loadData_mode.NONE*, *sampleRate=72000*)

Method to read sensor time series data from a .csv file.

Parameters

- **`mode`** (*`loadData_mode`*) – The path to the .csv file to read.
- **`mode`** – The measurement frequency.
- **`mode`** – Defines how to load the data.

Returns A pandas DataFrame representing the sensor data.

Return type pd.DataFrame

`anoog.io.csv_io.`**`read_csv_dataset`**`(`*datasetPath*, *csvName='capture.csv'*, *metaName='meta.yaml'*`)`

Loads a measurement dataset and meta-data.

Uses *`read_csv()`* and *`read_metadata()`* functions.

Parameters

- **`datasetPath`** (*`str`*) – The path to the dataset, to load it.
- **`csvName`** (*`str, optional`*) – The name of the measurement file.
- **`metaName`** (*`str, optional`*) – The name of the measurement metadata file.

Returns The measurement and the metadata of the drill.

Return type tuple of pd.DataFrame

`anoog.io.csv_io.`**`read_metadata`**`(`*yamlFile*`)`

Method to read meta data from a .yaml file.

Parameters **`yamlFile`** (*`str`*) – The path to the .yaml file to read.

Returns A pandas Series with the meta data information.

Return type pd.Series

4.2.2 anoog.io.data_io

This module used to load drill data. Internally this module uses the *`csv_io`* and makes the handling lot easier and with more features. The user can easily say which persons, with which feature-selection, with which feature-extraction and if there shoud be a split in train and test-data.

There are 2 main-functions (*`load_data()`* & `load_single_data()`), which use the other functions.

Author for `advanced_feature_extraction()`: Vadim Korzev Author for `feature_selection()` & `rename()`: Syon Kadkade Author for the rest: Tobia Ippolito

`anoog.io.data_io.`**`X_y_split`**`(`*data*`)`

Splits a DataFrame in X an the target column.

Parameters **`data`** (*`pd.DataFrame`*) – Data to split.

Returns Splittet data.

Return type tuple of pd.DataFrame and pd.Series

`anoog.io.data_io.`**`advanced_feature_extraction`**`(`*df*`)` → pandas.core.frame.DataFrame

Extracts special features like energy and resistens from timerow for every drill.

Parameters **`df`** (*`pd.DataFrame`*) – Data for extract the feature.

Returns Extracted Features.

Return type pd.DataFrame

`anoog.io.data_io.`**`extract_feature`**`(`*data*, *from_feature: str*, *function*`)`

Extracts a Feature of a drill timeseries data with the given function for every drill.

Parameters

- **`data`** (*`callable`*) – Data for extract the feature.
- **`from_feature`** (*`str`*) – The column, which will be used for extraction (‘Audio’, ‘Current’ or ‘Voltage’).

- **data** – Function for extract the feature (like np.mean).

Returns Feature list of the given function for every drill.

Return type list

`class anoog.io.data_io.`**`extraction_mode`**`(`*value*`)`

Bases: `enum.Enum`

An enumeration.

`MANUEL = 4`

`NONE = 1`

`TSFRESH = 3`

`TSFRESH_WITH_PARAMS = 2`

`anoog.io.data_io.`**`feature_extraction`**`(`*data*, *mode*, *data_path*, *show=True*`)` → pandas.core.frame.DataFrame

Applies feature-extraction on the data.

It's can be tsfresh or manuel and there are some more variations.

Parameters

- **data** (*pd.DataFrame*) – Data for extract the feature.
- **mode** (*extraction_mode*) – Feature-Extraction mode.
- **data_path** (*str*) – Path to the data.
- **show** (*bool, optional*) – Defines if the output of Tsfresh should be showed.

Returns Extracted Features.

Return type pd.DataFrame

`anoog.io.data_io.`**`feature_selection`**`(`*X*, *y*, *mode*, *data_path*`)` → pandas.core.frame.DataFrame

Applies feature-selection on the data.

Parameters

- **X** (*pd.DataFrame*) – Data for select the feature.
- **y** (*pd.Serie*) – Target-column.
- **mode** (*selection_mode*) – Feature-Selection mode.
- **data_path** (*str*) – Path to the data.

Returns Selected Features.

Return type pd.DataFrame

`anoog.io.data_io.`**`load_data`**`(`*data_path='../../../data/2021-11-09'*, *persons=['tippolit', 'vkorzev']*, *extraction=extraction_mode.MANUEL*, *show_extraction=False*, *selection=selection_mode.NONE*, *train_test_split=False*, *test_size=0.3*, *save_as_csv=False*, *csv_name=None*, *normalize=False*`)` → pandas.core.frame.DataFrame

Loads Data and prepares them. Feature Extraction and Feature Selection. Labeling and Resampling. Split in train/test or not.

All controllable over the arguments.

Parameters

- **data_path** (*str, optional*) – Path to the location of the data, which should be loaded.

- **persons** (*list of str, optional*) – Names of the persons who drilled (the data of these persons will be loaded).
- **extraction** (*extraction_mode*, optional) – Kind of extraction, which will be used for the timerows.
- **show_extraction** (*bool, optional*) – If Tsfresh selected as feature-extraction-method, the output of it will be visible or not.
- **selection** (*selection_mode*, optional) – Kind of selection, which will be used for the extracted features.
- **train_test_split** (*bool, optional*) – Defines if there should be a train-test-split
- **test_size** (*float, optional*) – Defines the size of the split, if train-test-split is setted on True (in percentage).
- **save_as_csv** (*bool, optional*) – Defines if the loaded data should be saved in a csv file.
- **csv_name** (*str, optional*) – If save_as_csv is True, this param defines, which name the file should have.
- **normlize** – Decided if the data should be normalized.

Returns The loaded data.

Return type pd.DataFrame

anoog.io.data_io.**load_features**(*data_path*, *name=None*)

Function to load the saved selected features from tsfresh.

Parameters

- **data_path** (*str*) – Path where the features are stored.
- **name** (*str, optional*) – Name of the file.

Returns Features from Tsfresh.

Return type dict

anoog.io.data_io.**load_single_data**(*person_id*, *person='tippolit'*, *data_path='../../../data/2021-11-09'*, *drill_id=0*, *extraction=extraction_mode.MANUEL*, *show_extraction=False*, *selection=selection_mode.NONE*, *train_test_split=False*, *test_size=0.3*, *save_as_csv=False*, *csv_name=None*, *normalize=False*) → pandas.core.frame.DataFrame

Loading a single drill-data-entry. The last drill in the folder.

There are many options to set.

Parameters

- **data_path** (*str, optional*) – Path to the location of the data, which should be loaded.
- **persons** (*list of str, optional*) – Names of the persons who drilled (the data of these persons will be loaded).
- **extraction** (*extraction_mode*, optional) – Kind of extraction, which will be used for the timerows.
- **show_extraction** (*bool, optional*) – If Tsfresh selected as feature-extraction-method, the output of it will be visible or not.
- **selection** (*selection_mode*, optional) – Kind of selection, which will be used for the extracted features.

- **train_test_split** (*bool, optional*) – Defines if there should be a train-test-split
- **test_size** (*float, optional*) – Defines the size of the split, if train-test-split is setted on True (in percentage).
- **save_as_csv** (*bool, optional*) – Defines if the loaded data should be saved in a csv file.
- **csv_name** (*str, optional*) – If save_as_csv is True, this param defines, which name the file should have.
- **normlize** – Decided if the data should be normalized.

Returns The loaded data.

Return type pd.DataFrame

anoog.io.data_io.**model_based_feature_selection**(*X*, *y*, *n=5*)
Get the n best Features elected by a RandomForest.

Parameters

- **X** (*pd.DataFrame*) – Data for select the feature.
- **y** (*pd.Serie*) – Target-column.
- **n** (*int, optional*) – Number of features, which will take n-best features.

Returns Returns new X with selected Features.

Return type pd.DataFrame

anoog.io.data_io.**remove_audio_features**(*data*) → pandas.core.frame.DataFrame
Removes all features with audio context.

Parameters **data** (*pd.DataFrame*) – Data for remove the audio features.

Returns Returns new data without audio features.

Return type pd.DataFrame

anoog.io.data_io.**rename**(*score*, *list_names*)
Masks all names, which have a True score.

Parameters

- **score** (*bool*) – Score of the features.
- **list_names** (*list of str*) – List with names of the features.

Returns New selected features.

Return type list of str

anoog.io.data_io.**resampling**(*data*) → pandas.core.frame.DataFrame
Resamples the data.

Scaled down the data-size (timerow) without loose to much informations.

Parameters **data** (*pd.DataFrame*) – The data which will be resampled.

Returns The down sampled data.

Return type pd.DataFrame

anoog.io.data_io.**save_data**(*data_path*, *datasets*, *name*)
Saves the data localy.

Parameters

- **data_path** (*str*) – Path to save the data.
- **datasets** (*list of list of pd.DataFrames/pd.Series*) – Data to save.
- **name** (*str*) – Name of the file to save the data.

anoog.io.data_io.**save_features**(*data_path*, *params: dict*)

Function to save the selected features from tsfresh.

Parameters

- **data_path** (*str*) – Path to save the features.
- **params** (*dict*) – Features von Tsfresh.

class anoog.io.data_io.**selection_mode**(*value*)

Bases: enum.Enum

An enumeration.

MODEL_BASED = 5

NONE = 1

SELECTKBEST = 3

TSFRESH = 2

VARIANCE = 4

anoog.io.data_io.**split_data**(*X*, *y*, *test_size_in_percent=0.3*, *should_split=False*) → list

Splits the data in train and testdata.

Wrapps the data in a list.

Parameters

- **X** (*pd.DataFrame*) – Data for select the feature.
- **y** (*pd.Serie*) – Target-column.
- **test_size_in_percent** (*float*) – Größe des splits/der Testdaten.
- **should_split** (*bool*) – Entscheidet, ob ein split der Daten vorgenommen werden soll.

Returns Returns the splitet data as diffrent datasets.

Return type list of list which contains X and y (maybe X_train, y_train,…)

anoog.io.data_io.**y_labeling**(*data*, *meta_data*, *extraction*)

Creates target-column from ID-column and persons.

Parameters

- **meta_data** (*pd.DataFrame*) – Data to get ID and operator.
- **extraction** (*extraction_mode*) – Kind of extraction, which will be used for the timerows.

Returns The target-column. Represents the operator.

Return type pd.Series

4.3 anoog.model

This Package has content about Machine Learning Algorithms.

4.3.1 anoog.model.model

This module is used to applie machine-learning algorithm.

Contains functions for train, predict and evaluate ai-model.

Author: Tobia Ippolito

`class anoog.model.model.MODELS`(*value*)

Bases: `enum.Enum`

An enumeration.

`ADA_BOOST = 4`

`KNN = 7`

`LOGISTIC_REGRESSION = 3`

`NAIVE_BAYES = 6`

`RANDOM_FOREST = 1`

`SVM = 2`

`VOTING_CLASSIFIER = 5`

`anoog.model.model.evaluate_model`(*model*, *X_test*, *y_test*, *X_train*, *normalize=False*)

Evaluats a model. Means predict new data and returns the accuracy and confusion-matrix. Can normalize the data.

Parameters

- **model** (*sklearn.base.BaseEstimator*) – Model used for evaluation.
- **X_test** (*pd.DataFrame*) – Data which will be predict.
- **y_test** (*pd.Series*) – Target value of the test-data.
- **X_train** (*pd.DataFrame*) – Data which will be used for normalization.
- **normalize** (*bool, optional*) – Defines if the data should be normalize (if True, uses X_train for normalization).

Returns A list of predicted classes with the 2 probabilities.

Return type tuple of accuracy and confusion matrix.

`anoog.model.model.evaluate_model_with_cross_validation`(*X*, *y*, *model=MODELS.RANDOM_FOREST*, *cv=5*)

Evaluats a model with cross-validation.

Parameters

- **X** (*pd.DataFrame*) – Features which will be splittet in train and validation datasets.
- **y** (*pd.Series*) – Target value of the data (which will be splittet in train and validation datasets).
- **model** (*MODELS*) – Model used for evaluation.

- **cv** (*int, optional*) – Defines the number of cross-validation-datasets.

Returns A list of predicted classes with the 2 probabilities.

Return type tuple of accuracy and confusion matrix.

`anoog.model.model.`**`get_most_important_features`**(*model*, *X*, *n=5*)

Calculates the n most important features of a RandomForestClassifier. Prints the result. See *get_most_important_features_as_list()* for getting a result returned.

Parameters

- **model** (*sklearn.base.BaseEstimator*) – Model used for feature-selection.
- **X** (*pd.DataFrame*) – Features which will be used for information-source by columns specifications.
- **n** (*int*) – Number of Features to select.

`anoog.model.model.`**`get_most_important_features_as_list`**(*model*, *X*, *n=5*)

Calculates the n most important features of a RandomForestClassifier.

Parameters

- **model** (*sklearn.base.BaseEstimator*) – Model used for feature-selection.
- **X** (*pd.DataFrame*) – Features which will be used for information-source by columns specifications.
- **n** (*int*) – Number of Features to select.

Returns Most important features.

Return type list of names of selected-features.

`anoog.model.model.`**`predict`**(*model*, *predict_data*, *X_train*, *normalize=False*)

Predicts new data with a model. Can normalize the data.

Parameters

- **model** (*sklearn.base.BaseEstimator*) – Model used for prediction.
- **predict_data** (*pd.DataFrame*) – Data which will be predict.
- **X_train** (*pd.DataFrame*) – Data which will be used for normalization.
- **normalize** (*bool, optional*) – Defines if the data should be normalize (if True, uses X_train for normalization).

Returns A list of predicted classes.

Return type numpy.ndarray

`anoog.model.model.`**`predict_proba`**(*model*, *predict_data*, *X_train*, *normalize=False*)

Predicts new data with a model in percent. Can normalize the data.

Parameters

- **model** (*sklearn.base.BaseEstimator*) – Model used for prediction.
- **predict_data** (*pd.DataFrame*) – Data which will be predict.
- **X_train** (*pd.DataFrame*) – Data which will be used for normalization.
- **normalize** (*bool, optional*) – Defines if the data should be normalize (if True, uses X_train for normalization).

Returns A list of predicted classes with the 2 probabilities.

Return type numpy.ndarray

`anoog.model.model.`**`train_adaboost`**(*X_train*, *y_train*, *auto_params=False*, *normalize=True*, *cv=3*)

Trains a AdaBoost with the given data.

Hyperparameters are configured by Tobia Ippolito. Alternativly GridSearch can be used. But there have to be enough data.

The traindata can be normalize.

Parameters

- **X_train** (*pd.DataFrame*) – Features which will be used for training.
- **y_train** (*pd.Series*) – Target-feature which will be used for training.
- **auto_params** (*bool, optional*) – Defines whether or not using preconfigured hyperparameter or lets GridSearch tunes the model.
- **normalize** (*bool, optional*) – Defines if the data should be normalized.
- **cv** (*int, optional*) – Defines the number of cross-validation-datasets by auto variable = True.

Returns Returns the trained classifier.

Return type sklearn.ensemble.AdaBoostClassifier

`anoog.model.model.`**`train_knn`**(*X_train*, *y_train*, *auto_params=False*, *normalize=True*, *cv=3*)

Trains a K-Nearest Neighbors with the given data.

Hyperparameters are configured by Tobia Ippolito. Alternativly GridSearch can be used. But there have to be enough data.

The traindata can be normalize.

Parameters

- **X_train** (*pd.DataFrame*) – Features which will be used for training.
- **y_train** (*pd.Series*) – Target-feature which will be used for training.
- **auto_params** (*bool, optional*) – Defines whether or not using preconfigured hyperparameter or lets GridSearch tunes the model.
- **normalize** (*bool, optional*) – Defines if the data should be normalized.
- **cv** (*int, optional*) – Defines the number of cross-validation-datasets by auto variable = True.

Returns Returns the trained classifier.

Return type sklearn.neighbors.KNeighborsClassifier

`anoog.model.model.`**`train_logistic_regression`**(*X_train*, *y_train*, *auto_params=False*, *normalize=True*, *cv=3*)

Trains a Logistic Regression with the given data.

Hyperparameters are configured by Tobia Ippolito. Alternativly GridSearch can be used. But there have to be enough data.

The traindata can be normalize.

Parameters

- **X_train** (*pd.DataFrame*) – Features which will be used for training.
- **y_train** (*pd.Series*) – Target-feature which will be used for training.

- **auto_params** (*bool, optional*) – Defines whether or not using preconfigured hyperparameter or lets GridSearch tunes the model.
- **normalize** (*bool, optional*) – Defines if the data should be normalized.
- **cv** (*int, optional*) – Defines the number of cross-validation-datasets by auto variable = True.

Returns Returns the trained classifier.

Return type sklearn.linear_model.LogisticRegression

anoog.model.model.**train_naive_bayes**(*X_train*, *y_train*, *auto_params=False*, *normalize=True*, *cv=3*)

Trains a Naive Bayes with the given data.

Hyperparameters are configured by Tobia Ippolito. Alternativly GridSearch can be used. But there have to be enough data.

The traindata can be normalize.

Parameters

- **X_train** (*pd.DataFrame*) – Features which will be used for training.
- **y_train** (*pd.Series*) – Target-feature which will be used for training.
- **auto_params** (*bool, optional*) – Defines whether or not using preconfigured hyperparameter or lets GridSearch tunes the model.
- **normalize** (*bool, optional*) – Defines if the data should be normalized.
- **cv** (*int, optional*) – Defines the number of cross-validation-datasets by auto variable = True.

Returns Returns the trained classifier.

Return type sklearn.naive_bayes.GaussianNB

anoog.model.model.**train_random_forest**(*X_train*, *y_train*, *auto_params=False*, *normalize=False*, *cv=3*)

Trains a RandomForrest with the given data.

Hyperparameters are configured by Tobia Ippolito. Alternativly GridSearch can be used. But there have to be enough data.

The traindata can be normalize.

Parameters

- **X_train** (*pd.DataFrame*) – Features which will be used for training.
- **y_train** (*pd.Series*) – Target-feature which will be used for training.
- **auto_params** (*bool, optional*) – Defines whether or not using preconfigured hyperparameter or lets GridSearch tunes the model.
- **normalize** (*bool, optional*) – Defines if the data should be normalized.
- **cv** (*int, optional*) – Defines the number of cross-validation-datasets by auto variable = True.

Returns Returns the trained classifier.

Return type sklearn.ensemble.RandomForestClassifier

anoog.model.model.**train_svc**(*X_train*, *y_train*, *auto_params=False*, *normalize=True*, *cv=3*)

Trains a SupportVectorMachine with the given data.

Hyperparameters are configured by Tobia Ippolito. Alternativly GridSearch can be used. But there have to be enough data.

The traindata can be normalize.

Parameters

- **X_train** (*pd.DataFrame*) – Features which will be used for training.
- **y_train** (*pd.Series*) – Target-feature which will be used for training.
- **auto_params** (*bool, optional*) – Defines whether or not using preconfigured hyperparameter or lets GridSearch tunes the model.
- **normalize** (*bool, optional*) – Defines if the data should be normalized.
- **cv** (*int, optional*) – Defines the number of cross-validation-datasets by auto variable = True.

Returns Returns the trained classifier.

Return type sklearn.svm.SVC

anoog.model.model.**train_voting_classifier**(*X_train*, *y_train*, *auto_params=False*, *normalize=True*, *cv=3*)

Trains a Voting Classifier with the given data.

Hyperparameters are configured by Tobia Ippolito. Alternativly GridSearch can be used. But there have to be enough data.

The traindata can be normalize.

Parameters

- **X_train** (*pd.DataFrame*) – Features which will be used for training.
- **y_train** (*pd.Series*) – Target-feature which will be used for training.
- **auto_params** (*bool, optional*) – Defines whether or not using preconfigured hyperparameter or lets GridSearch tunes the model.
- **normalize** (*bool, optional*) – Defines if the data should be normalized.
- **cv** (*int, optional*) – Defines the number of cross-validation-datasets by auto variable = True.

Returns Returns the trained classifier.

Return type sklearn.ensemble.VotingClassifier

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX