KI & Maschinenlernen auf dem Mikrocontroller

Teil I: Klassisches ML auf dem Arduino Uno

Urheber: Thomas Jörg, thomas@iludis.de

Stand 10. April 2023

Abb. 2 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „a cute little robot during parachute jump with white background, digital art“ von Jörg [CC BY-SA 4.0 International]

Abb. 1 Eigene Fotos TinyML Kit

Klassisches Maschinenlernen auf Mikrocontrollern 3
Was bedeutet ‚klassisches ML'? 3
a) EM-Learn Installation 4
b) Micromlgen / Eloquent Arduino Installation 6
Überblick und Vergleich der beiden Bibliotheken: 8
Installation 8
Der Prozess von den Daten zum Arduino 9
Aufbau des Arduino-Systems 9
Daten Sammeln! 10
Minimalistisches Skript zum Datensammeln, ohne OLED-Display 11
Klassifikations-Modelle mit Python 14
Hinweis: Alle Skripte in diesem Dokument sind getestet (19.10.2022) 14
Eloquent: Decision-Tree-Klassifikation ohne Normalisierung 15
Eloquent: Arduino-Code für den DecisionTree ohne Normalisierung 16
Eloquent: Decision-Tree-Klassifikation mit Normalisierung 17
Eloquent: Arduino-Code für DecisionTree **mit** Normalisierung 19
Eloquent: RandomForest-Klassifikation ohne Normalisierung 20
Eloquent: XGBoost-Klassifikation mit 3 Bäumen ohne Normalisierung 21
Eloquent: Naive-Bayes-Klassifikation mit Normalisierung 22
Eloquent: Naive-Bayes-Klassifikation ohne Normalisierung 23
Eloquent: SupportVectorMachine-Klassifikation ohne Normalisierung 25
Eloquent: Regressions-Modelle 27
Eloquent: Principal Component Analysis, PCA 28
Eloquent: Arduino-Quellcode, der PCA und Decision-Tree verbindet 30
EM-Learn: Decision-Tree-Klassifikation ohne Normalisierung 31
EM-Learn: Arduino-Code für den DecisionTree ohne Normalisierung 32
EM-Learn: Random-Forest-Klassifikation ohne Normalisierung 33
EM-Learn: NeuralNet-Klassifikation mit Normalisierung 34
EM-Learn: Arduino-Code für das Neuronale Netz mit Normalisierung 35
EM-Learn: NaiveBayes-Klassifikation mit Normalisierung 37
EM-Learn: Arduino-Code für das Neuronale Netz mit Normalisierung 38

Klassisches Maschinenlernen auf Mikrocontrollern

Abb. 3 Bild [Gemeinfrei] erzeugt mit Stable Diffusion (lokale Installation); Prompt „Robot arm picking strawberries from plate, high end 3d rendering, digital art“ von Jörg [CC BY-SA 4.0 International]

Was bedeutet ‚klassisches ML‘?

Unter ‚klassischen‘ ML-Algorithmen versteht man im Allgemeinen alles außer DeepLearning, also unter anderem:

- DecisionTree (Entscheidungsbäume)
- Random Forests
- kNN (k-Nächste-Nachbarn)
- SVM, SupportVectorMachines (Stützvektormaschinen)
- Naive Bayes Klassifizierer
- *Multilayer-Perceptronen (sehr nahe an DeepLearning)*
- PCA (Principle Component Analysis)
- k-Means Clustering und EllipticEnvelops (hauptsächlich für Anomalie-Detection)

Bis auf die Multilayer-Perceptronen sind alle Algorithmen sehr schlank und lassen sich meistens problemlos auch auf kleinere Mikrocontroller (z.B. Arduino Uno) übertragen. Der Weg ist hierbei immer gleich:

1. Daten in Python importieren,
   notwendige und grundlegende Bibliotheken sind diese drei: Pandas, Numpy, Sklearn
2. Mittels Sklearn das jeweilige Modell trainieren.
3. Das fertige Modell noch im Pythonskript mittels spezieller Bibliotheken in C-Code konvertieren
4. Den C-Code als #include-Datei in ein Arduino-Template dazuladen
5. Das Modell nutzen.

Achtung, NORMALISIERUNG!

Einige Modelle benötigen zuvor eine Normalisierung der Eingabedaten. Dies geschieht innerhalb von Python oftmals mit dem MinMax-Scaler. Diesen Normalisierungs-Algorithmus wird in diesem Tutorial ausschließlich genutzt, weil er sehr einfach anzuwenden ist. Allerdings muss man diese Normalisierung dann ebenfalls ins Arduino-C mit übernehmen. Wo das nötig ist, wird aber bei den entsprechenden Beispielen erläutert.

All diese Algorithmen findet man auf zwei OpenSource-Bibliotheken verteilt, die man sich aus dem Netz leicht herunterladen und installieren kann. EMLearn und Eloquent/Micromlgen. Eloquent wird dabei in der letzten Zeit intensiv weiterentwickelt!

Beide Bibliotheken haben jeweils Vor- und Nachteile:

- EMLearn benötigt zwingend eine voll installierte Python-Umgebung und zusätzlich die **`Microsoft Build Tools.`** Man braucht anschließend keine zusätzliche Arduino-Libraries mehr, die erstellte Header-Datei lässt sich direkt in den Arduino-Ordner kopieren und in den C-Quellcode einbinden.

- Eloquent/MicroMLgen: Die Python-Bibliothek **`Micromlgen`** lässt sich auch auf einer portablen Python- oder Jupyter-Variante installieren wie zum Beispiel auf Edupyter von https://www.portabledevapps.net/
  Man braucht anschließend keine zusätzliche Arduino-Libraries mehr, die erstellte Header-Datei lässt sich direkt in den Arduino-Ordner kopieren und in den C-Quellcode einbinden.

SKLEARN-PORTER: Warum nicht?

Das Tool wird nicht weitergepflegt und erzeugt Fehlermeldungen, weil es mit den aktuellen Python-Versionen nicht mehr kompatibel ist. Schade.

a) # EM-Learn Installation

https://github.com/emlearn/emlearn
https://emlearn.readthedocs.io/en/stable/

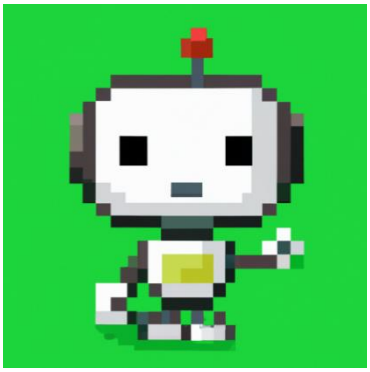


Abb. 4 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „a cute little robot learning, green background, pixel art“ von Jörg [CC BY-SA 4.0 International]

Umfasst die folgenden ML-Algorithmen:

1. Gaussian Naive Bayes
2. Decision Trees
3. Random Forests
4. EllipticEnvelops
5. Multilayer Perceptrons

Prozedere:

EMLearn benötigt zwingend die Microsoft Build Tools 2019 (siehe u.a. die Installationsanleitung für EdgeImpulse):

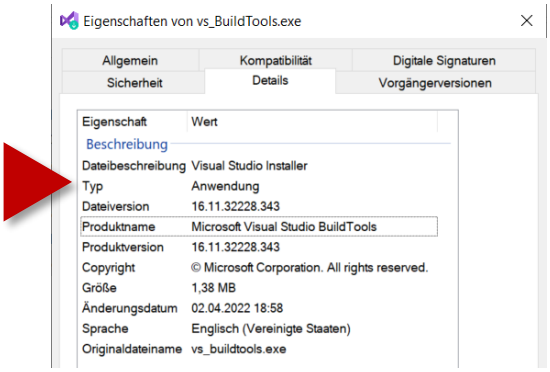


Abb. 5 Eigenes Screenshot

Es muss folgende Option ausgewählt sein: „Desktopentwicklung mit C++“

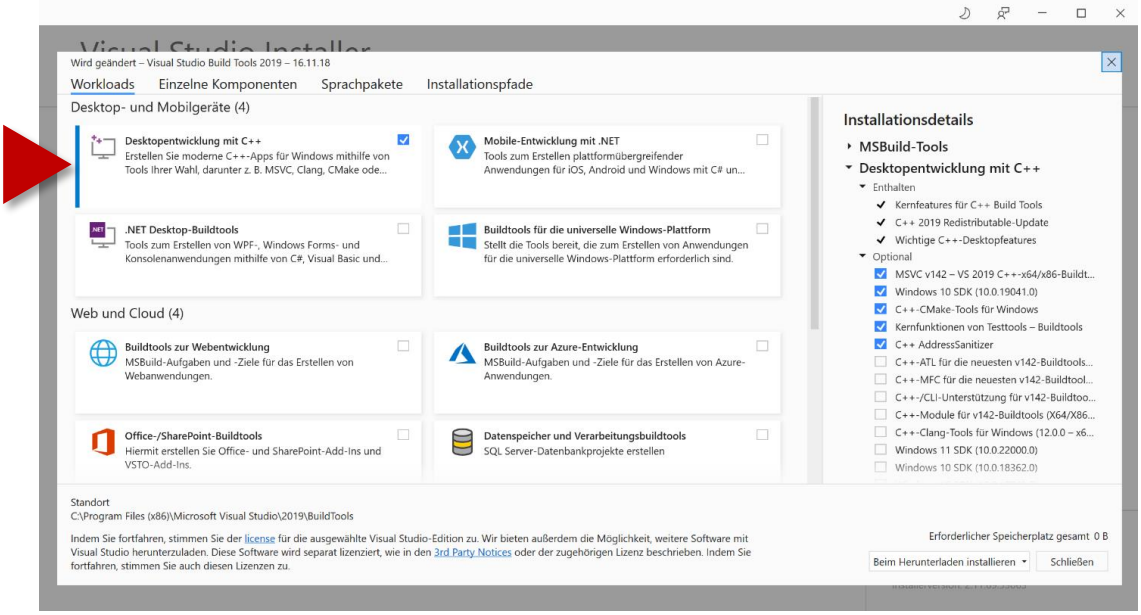


Abb. 6 Eigenes Screenshot

Weiterhin muss Anaconda installiert sein, emLearn funktioniert NICHT mit einer portablen Pythonversion!
Ist Anaconda installiert, öffnet man den Anaconda Prompt und gibt ein:

```
pip install --user emlearn
```

Abb. 7 Eigenes Screenshot

Anschließend startet man aus dem Anaconda Navigator ein Jupyter-Notebook:

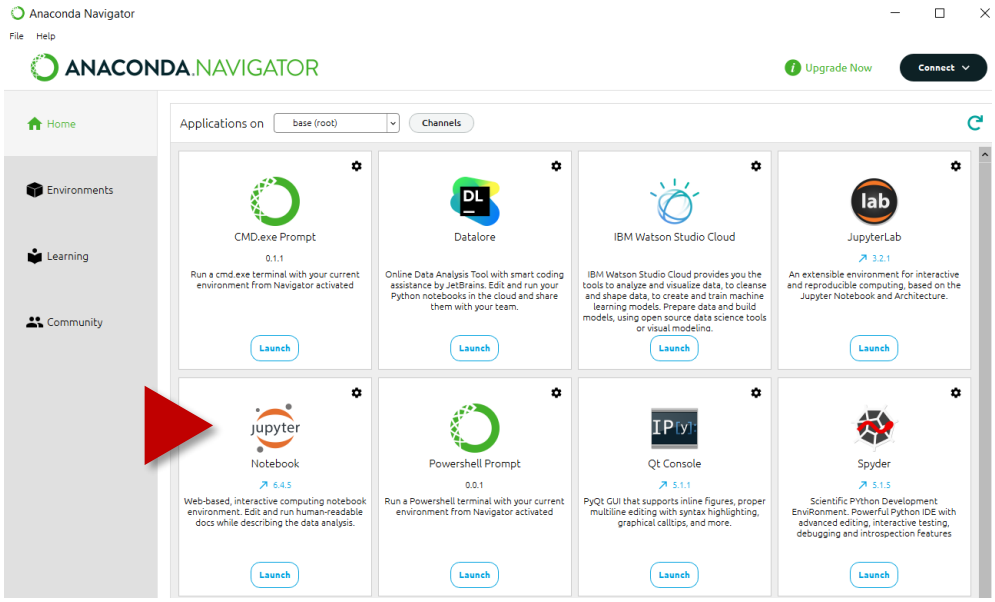


Abb. 8 Eigenes Screenshot

Im Jupyter-Notebook versucht man, EMLearn zu laden:

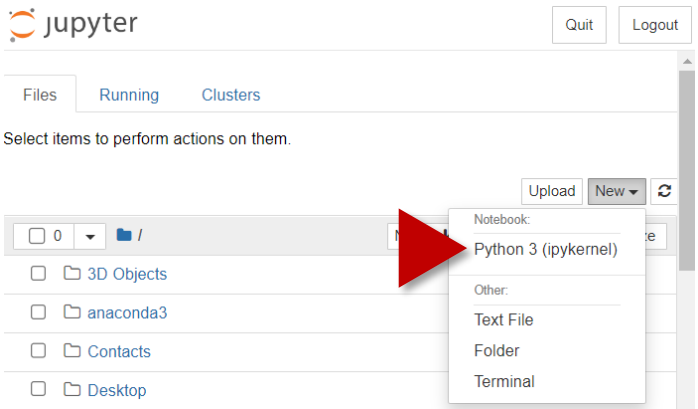


Abb. 9 Eigenes Screenshot

… und zwar mit folgender Befehlszeile:

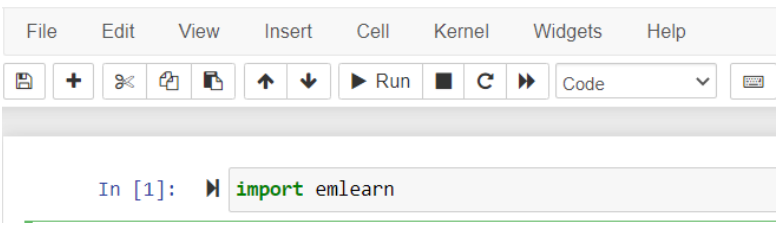


Abb. 10 Eigenes Screenshot

Wenn das fehlerfrei funktioniert, ist man fertig.
EMLearn benötigt keine weiteren Arduino-Bibliotheken!

b) Micromlgen / Eloquent Arduino Installation

https://github.com/eloquentarduino/micromlgen

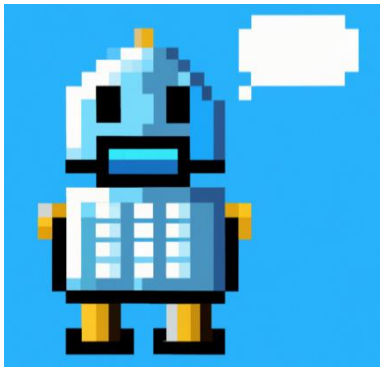


Abb. 11 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „a cute little robot speaking, blue background, pixel art“ von Jörg [CC BY-SA 4.0 International]

Umfasst die folgenden ML-Algorithmen:

- Gaussian Naïve Bayes
- DecisionTree
- RandomForest
- XGBoost
- Support Vector Machines (SVC and OneClassSVM)
- Relevant Vector Machines (from skbayes.rvm_ard_models package)
- SEFR
- PCA

Installation

Aus Gründen der Einfachheit gehen wir von einer portablen Installation aus, die zum Beispiel über Edupyter erfolgen kann:

https://www.portabledevapps.net/edupyter.php

Die Installation sollte nicht in den von Eduypter vorgeschlagenen Standardordner erfolgen, sondern in einen selbst gewählten Ordner, wie zum Beispiel auf einem USB-Stick oder einem externen Laufwerk:

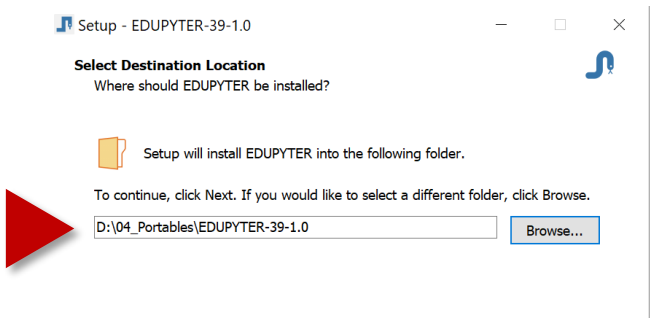


Abb. 12 Eigenes Screenshot

Startet man die Entwicklungsumgebung, erscheint ein Icon im Systray:

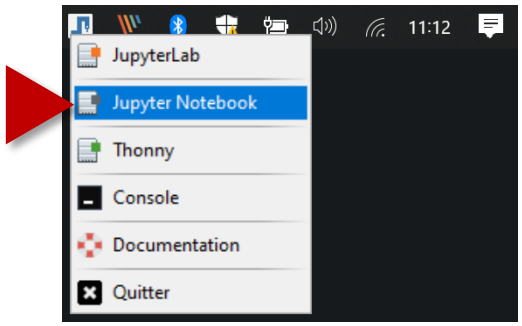


Abb. 13 Eigenes Screenshot

Hier startet man ein Jupyter-Notebook; es sollte sich ein Browser öffnen, in dem das Jupyter-Notebook läuft:

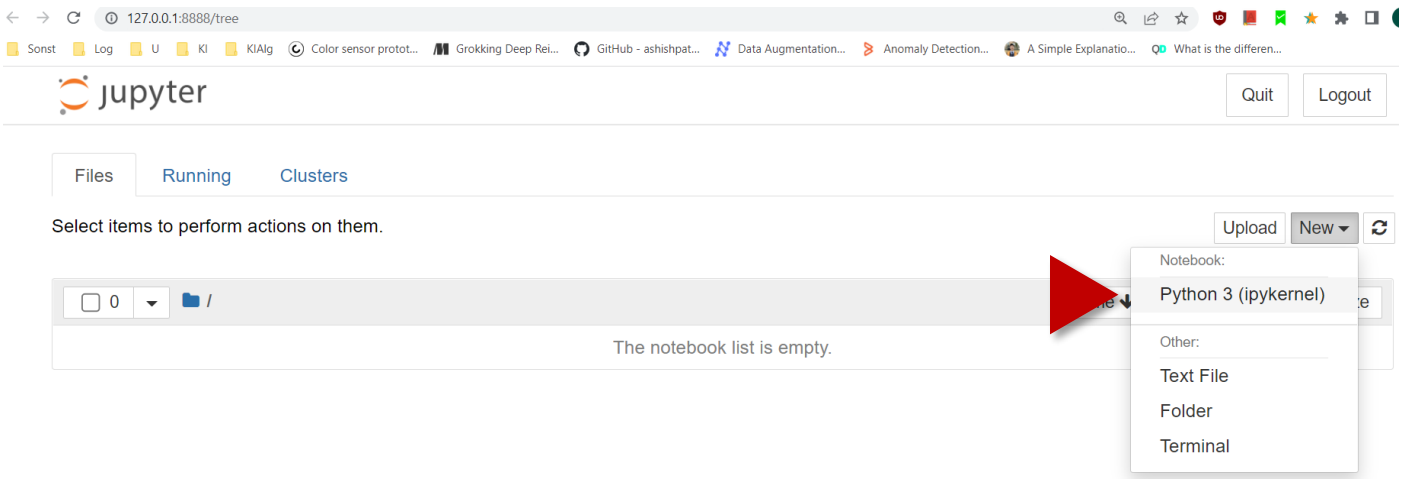


Abb. 14 Eigenes Screenshot

Innerhalb des neu erstellten jupyter-Notebooks gibt man nun den folgenden Befehl ein und führt in aus,

- entweder mit einem Klick auf ‚Run‘
- oder mit dem gleichzeitigen Drücken der Tasten ‚Strg‘ und ‚Enter‘

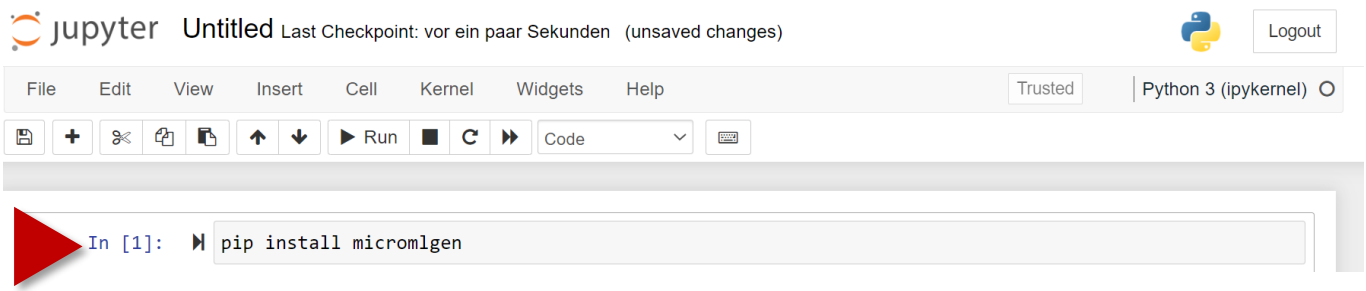


Abb. 15 Eigenes Screenshot

Anschließend kann man ab nun immer die micromlgen-Installation verwenden, sie ist auch nach dem nächsten Neustart persistent vorhanden.

Überblick und Vergleich der beiden Bibliotheken:

Stellt man beide Bibliotheken gegenüber, so zeichnen sich die Einsatzmöglichkeiten deutlicher ab:

	EMLearn	Micromlgen/Eloquent
DecisionTrees	Ja	Ja
Random Forests	Ja	Ja
kNN *	Nein	Nein
SVM	Nein	Ja
Naive Bayes	Ja	Ja
Perceptrons/NNets	Ja	Nein
PCA	Nein	Ja
k-Means Clustering	Nein	Nein
Elliptic Envelops	Ja	Nein
XGBoost	Nein	Ja
Portable Installation möglich?	*Nein*	*Ja*

**Hinweis zu kNN: Da der kNN-Algorithmus kein portables Modell durch Training erzeugt, sondern von jedem neuen Messpunkt den Abstand – ob nun geometrisch oder anderweitig – von den Punkten eines gelabelten Datensatzes berechnet, folgt hieraus: Der kNN benötigt jederzeit den vollen gelabelten Original-Datensatz innerhalb seines Speichers, wodurch sich Speicherprobleme ergeben. Ob ein kNN lauffähig ist oder nicht, hängt damit an der Größe des Datensatzes. Dennoch: Eine mögliche Bibliothek findet sich hier:* https://www.arduino.cc/reference/en/libraries/arduino_knn/

Der hauptsächliche Vorteil der EMLearn-Bibliothek liegt in der Implementierung eines Neuronalen Netzes. Ansonsten kann man auf diese Bibliothek verzichten.

C) A generalized package for porting ML models to C/Javascript/PHP/Micro-Python:

Achtung: zum aktuellen Zeitpunkt funktioniert das Paket nicht, weil einige Dependencies wohl inkompatibel sind. Der Entwickler ist informiert.

Diese Webseite wird laufend weiterentwickelt und deshalb kann nur ‚grob' auf die Inhalte eingegangen werden:

https://eloquentarduino.com/

https://github.com/eloquentarduino/everywhereml

Installation

Die Installation läuft analog der MicroMLgen/Eloquent-Prozedur mittels des folgenden pip-Befehls:

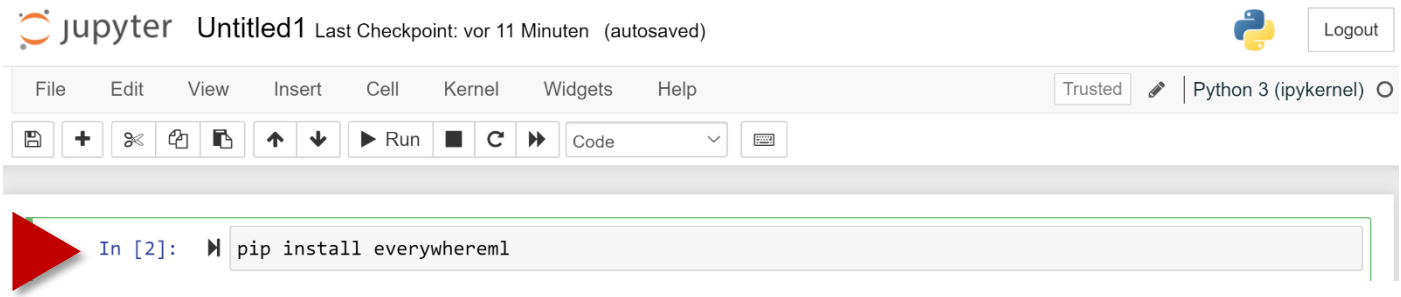


Abb. 16 Eigenes Screenshot

Der Prozess von den Daten zum Arduino

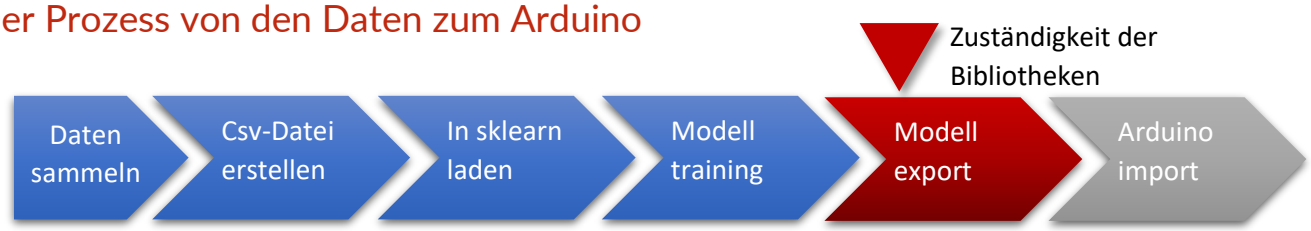


Abb. 17 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „a assembly line in a factory produces cookies, high end 3d rendering“ von Jörg [CC BY-SA 4.0 International]

Die blauen Bestandteile des Prozesses funktionieren wie ‚übliches‘ Maschinenlernen, werden also innerhalb der Pythonskripte – meist innerhalb von jupyter-Notebooks - durchgeführt.

Der rote Teil „Modell export“ ist die Aufgabe der drei Bibliotheken EMlearn / MicroMLgen / EverywhereML. Hier wird das Modell in die jeweilige Programmiersprache umgewandelt:

im Falle von Arduino-Code in C++,
im Falle von EverywhereML in einige weitere Programmiersprachen wie Javascript oder PHP.

Ein großer didaktischer Wert liegt darin, dass ein fertig trainiertes Modell sich als normaler C-Quellcode beschreiben und deshalb auch interpretieren lässt.

Es ‚demystifiziert‘ den abstrakt und ungreifbar erscheinende Begriff des Maschinenlern-Modells.

Aufbau des Arduino-Systems

Der hier verwendete Aufbau ist lediglich ein Beispiel. Er soll klar und einfach strukturiert und leicht nachvollziehbar sein. Auch die 3D-gedruckten Teile sind für die Durchführung nicht notwendig, aber sie unterstützen die Klarheit.

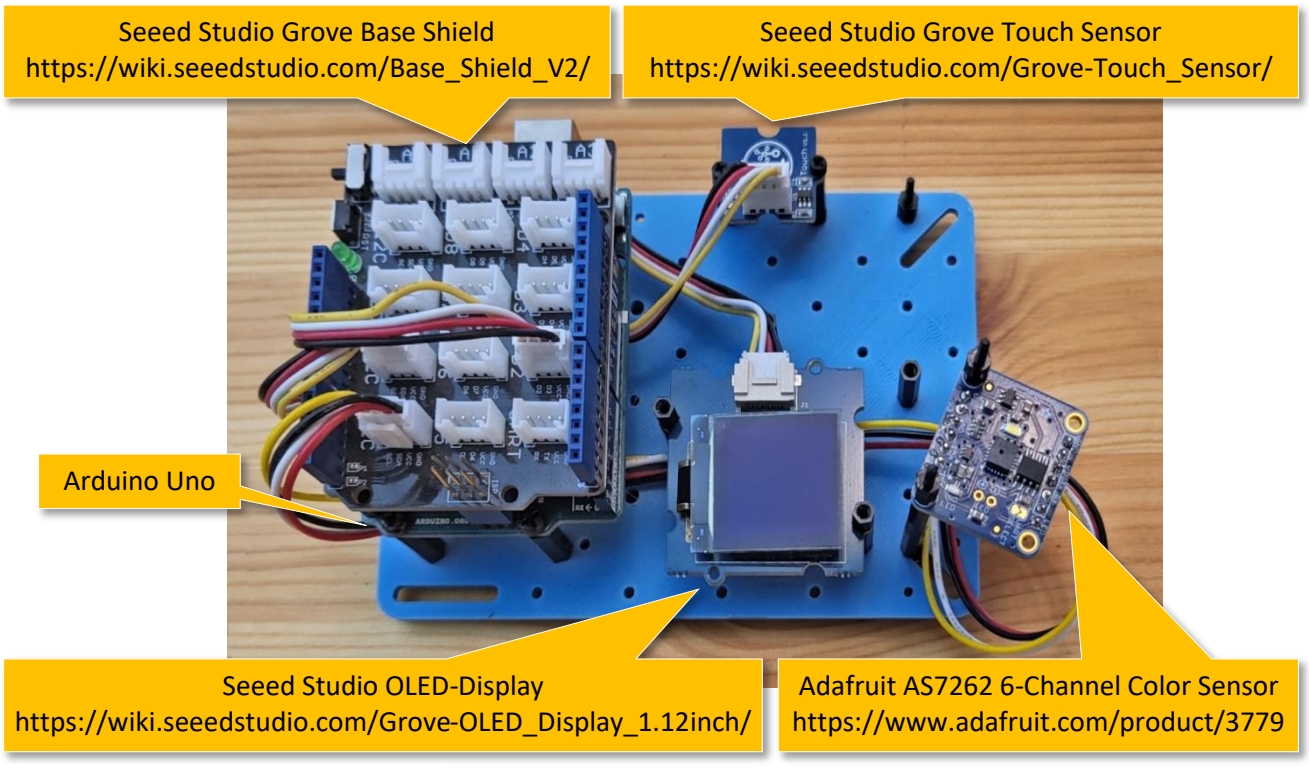


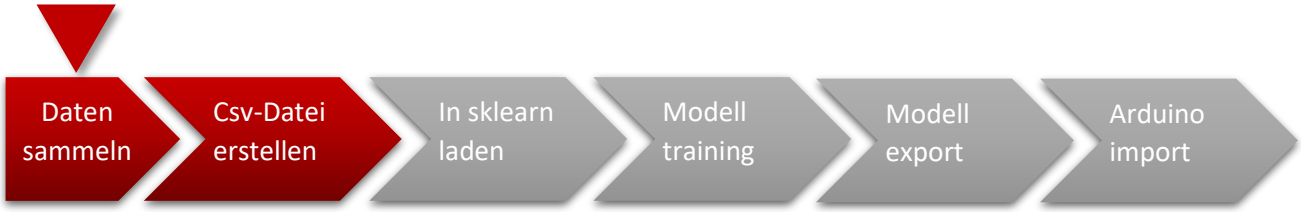
Abb. 18 Eigenes Foto

Daten Sammeln!

Grundsätzlich gilt: Besser man nimmt mit demselben Hardware-System die Daten auf, auf dem später auch das fertige Vorhersagemodell laufen wird. Denn nur dann ist sichergestellt, dass die Sensorwerte der Inferenzdatensätze auch gleich denjenigen der Trainingsdatensätze sind.

Hier sollen verschiedenfarbige Legobausteine verwendet werden, weil deren Farbwerte sehr leicht zu messen sind. Außerdem kennt man aus anderen Automatisierungs-Szenarien die Aufteilung von Legobausteinen nach Farben bereits – wodurch sich gute Vergleichswerte ergeben.

Abb. 19 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „little robot with different lego pieces, high quality 3d rendering“ von Jörg [CC BY-SA 4.0 International]



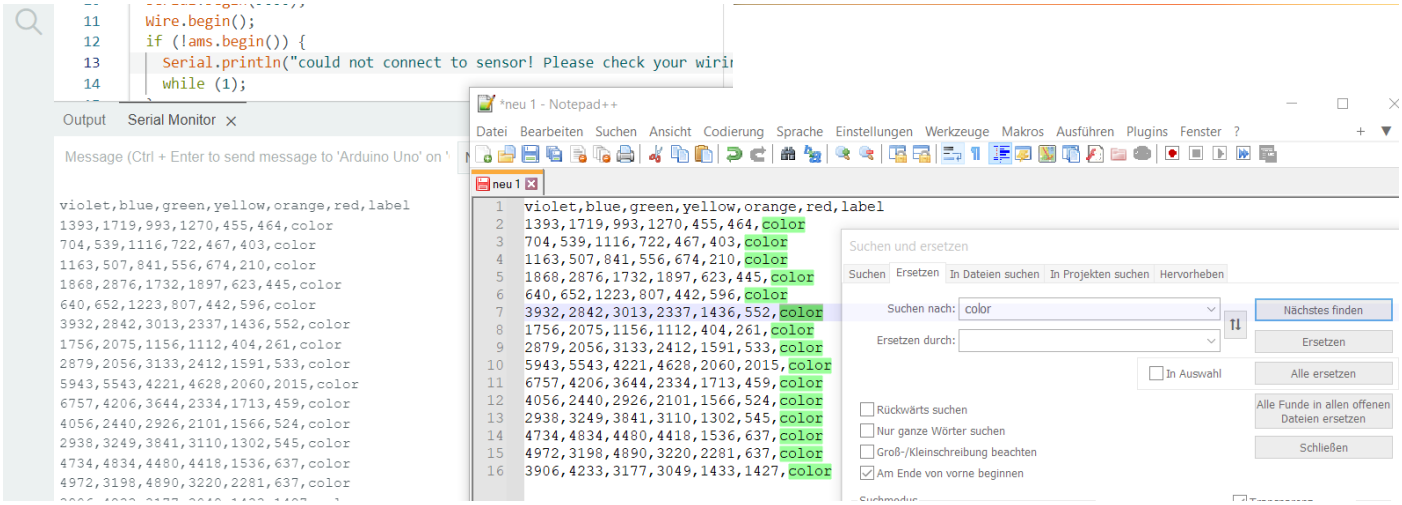


Abb. 20 Eigenes Screenshot

Vorschlag für die Labelcodierung der Farben:

Rot: 1	Orange: 2	Gelb: 2
Grün: 3	Türkis: 4	Blau: 5
Violett: 6	Nichts: 0	

Nachdem man alle Daten untereinander eingefügt, und die Labels definiert hat, kann man die Datei als csv-Datei abspeichern.

Minimalistisches Skript zum Datensammeln, ohne OLED-Display

Um einen Datensatz aufzunehmen, drückt man auf einen Knopf. Dieser ist an PIN 2 angeschlossen. Bei Knopfdruck wird dann die Beleuchtungs-LED des Sensors angeschaltet, ein Datenpunkt aufgenommen und in einem Array gespeichert und zuletzt wird die Mess-LED wieder abgeschaltet. Der Knopfdruck ist als kleine State-Machine umgesetzt; ein neuer Datenpunkt kann erst dann wieder aufgenommen werden, wenn der Knopf losgelassen wurde.

Abb. 21 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „cute little robot with a wicker basket bends down to pick strawberries, polaroid photo“ von Jörg [CC BY-SA 4.0 International]

```
#include <Wire.h>
#include "Adafruit_AS726x.h"

Adafruit_AS726x ams;
uint16_t sensorValues[AS726x_NUM_CHANNELS];
bool button = false;
bool altbutton = false;
void setup() {
  pinMode(2, INPUT);
  Serial.begin(9600);
  Wire.begin();
  if (!ams.begin()) {
    Serial.println("could not connect to sensor!");
    while (1);
  }
  Serial.println("violet,blue,green,yellow,orange,red,label");
}

void loop() {
  button = digitalRead(2);
  if (button) {
    if (altbutton != button) {
      altbutton = button;
      ams.drvOn();  //uncomment this if you want to use the driver LED for readings
      ams.startMeasurement();
      bool rdy = false;
      while (!rdy) {
        delay(5);
        rdy = ams.dataReady();
      }
      ams.readRawValues(sensorValues);
      for (int i = 0; i < 6; i++) {
        Serial.print(sensorValues[i]);
        Serial.print(",");
      }
      Serial.println("color");
      ams.drvOff();
    }
  }
  if (!button) {altbutton = false;}
}
```

Leicht erweitertes Skript zum Datensammeln, mit OLED-Display


```
#include <Wire.h>
#include "Adafruit_AS726x.h"
#include <SeeedGrayOLED.h>

Adafruit_AS726x ams;
uint16_t sensorValues[AS726x_NUM_CHANNELS];
char *farben[] = { "vio:", "blu:", "grn:", "yel:", "ora:", "red:" };
bool button = false;
bool altbutton = false;
void setup() {
  pinMode(2, INPUT);
  Serial.begin(9600);
  Wire.begin();
  if (!ams.begin()) {
    Serial.println("could not connect to sensor! Please check your wiring.");
    while (1)
      ;
  }
  SeeedGrayOled.init(SH1107G);
  SeeedGrayOled.clearDisplay();
  SeeedGrayOled.setNormalDisplay();
  SeeedGrayOled.setVerticalMode();
  SeeedGrayOled.setTextXY(0, 0);
  SeeedGrayOled.putString("Farbwerte:");
  for (int i = 0; i < 6; i++) {
    SeeedGrayOled.setTextXY(i + 1, 0);
    SeeedGrayOled.putString(farben[i]);
  }
  Serial.println("violet,blue,green,yellow,orange,red,label");
}

void loop() {
  button = digitalRead(2);
  if (button) {
    if (altbutton != button) {
      altbutton = button;
      ams.drvOn();  //uncomment this if you want to use the driver LED for readings
      ams.startMeasurement();
      bool rdy = false;
      while (!rdy) {
        delay(5);
        rdy = ams.dataReady();
      }
      ams.readRawValues(sensorValues);
      for (int i = 0; i < 6; i++) {
        Serial.print(sensorValues[i]);
        Serial.print(",");
      }
      Serial.println("color");

      for (int i = 0; i < 6; i++) {
        SeeedGrayOled.setTextXY(i + 1, 40);
        SeeedGrayOled.putString("       ");
        SeeedGrayOled.setTextXY(i + 1, 40);
        SeeedGrayOled.putNumber(sensorValues[i]);
      }
      ams.drvOff();
    }
  }
  if (!button) {
    altbutton = false;
  }
}
```

Inhalt der Datei “LegoTraining.csv“

```
violet,blue,green,yellow,orange,red,label
6436,2398,3085,1785,2366,795,2
5289,2528,2930,1145,1225,348,2
7155,4470,3015,3809,1982,1437,2
6775,3560,2739,1604,1591,520,2
3513,1968,2001,1216,1307,525,2
5870,2486,2469,1084,1343,407,2
5680,2398,4918,2668,2729,431,2
3502,1741,1330,1500,771,580,2
5944,2721,1689,1526,1184,583,2
3809,1915,4270,2974,3050,726,2
4562,2259,2332,1298,1131,517,2
9954,4135,4632,3971,3045,693,2
3623,1964,2756,2011,1542,487,2
1155,941,1752,1574,2068,1265,0
1155,942,1752,1574,2068,1265,0
1156,942,1752,1575,2068,1265,0
1156,943,1753,1576,2070,1266,0
1156,942,1752,1574,2070,1264,0
1156,942,1751,1573,2068,1263,0
1155,940,1751,1573,2067,1263,0
1156,942,1752,1574,2069,1263,0
1157,941,1753,1574,2069,1264,0
1157,941,1752,1574,2069,1264,0
1156,943,1753,1575,2071,1265,0
1455,2070,12829,15600,16553,6342,4
1901,3471,6188,7418,4757,2740,4
1202,1904,9822,11557,9058,4563,4
697,1475,7386,8692,7848,4258,4
923,1521,6790,8670,7852,4505,4
2073,2107,9749,10555,15562,6545,4
2424,2596,8504,10306,12084,6058,4
799,1879,7795,9142,11846,5299,4
1588,1578,4174,5406,5902,3111,4
1071,2716,13119,14168,19199,8464,4
1478,2500,8990,10577,12096,5422,4
1330,1359,4390,5723,7056,3309,4
2813,1689,3606,3894,5073,3980,1
2178,914,1447,1719,2025,2405,1
1349,663,1644,1892,1851,2565,1
1193,573,1326,1223,1703,2342,1
1471,674,1995,1312,3960,3230,1
1246,745,1043,1177,1601,2488,1
1200,709,1108,1311,2127,3714,1
1078,920,2042,1295,2376,3214,1
1596,813,1561,1694,2461,2751,1
1402,793,1712,1319,2326,2899,1
837,1547,4413,2979,2116,435,3
1206,2229,3356,4272,3322,2634,3
735,2518,1500,1248,966,854,3
1451,1815,4783,1955,3249,434,3
814,2308,3059,1813,1541,476,3
900,1860,3785,2042,990,426,3
518,1459,3014,1683,1235,453,3
2558,5382,2274,2344,1100,2405,3
690,1704,1933,1412,970,450,3
900,1382,2556,1948,1603,476,3
1664,2611,2620,2185,1881,793,3
```

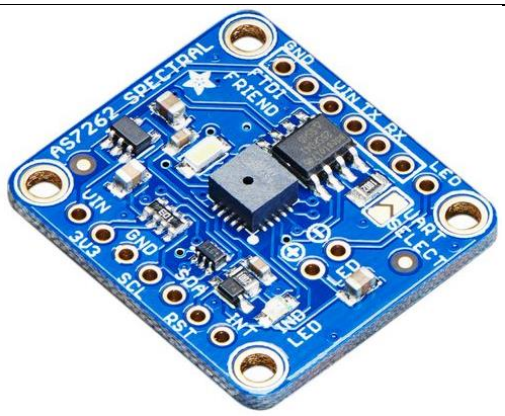


Abb. 22 Eigenes Foto

AS7262 6-Kanal-Spektralsensor.
Werte werden als rohe 16-Bit-Werte ausgelesen.

Klartext-Labels werden durch Zahlenwerte codiert:
0: Nothing
1: Rot
2: Orange
3: Grün
4: Türkis

Klassifikations-Modelle mit Python

Die Daten warden zunächst in ein skLearn-Pythonskript geladen, gegebenenfalls normalisiert und dann anschließend das Modell trainiert. Das fertige Modell wird dann im selben Skript in C-Code umgewandelt. Dabei kann man den fertigen Quellcode entweder direkt aus der Ausgabe herauskopieren und in die Arduino-Entwicklungsumgebung einfügen. Oder man nutzt die gleichzeitig mit abgespeicherte C-Header-Datei und kopiert diese in den Arduino-Ordner mit den anderen C-Quellcode-Dateien.

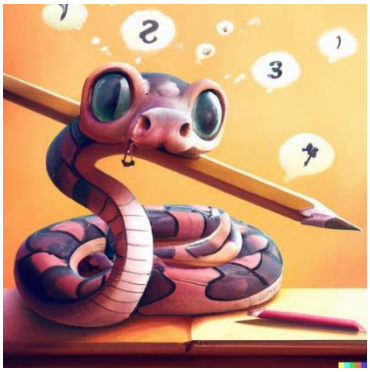
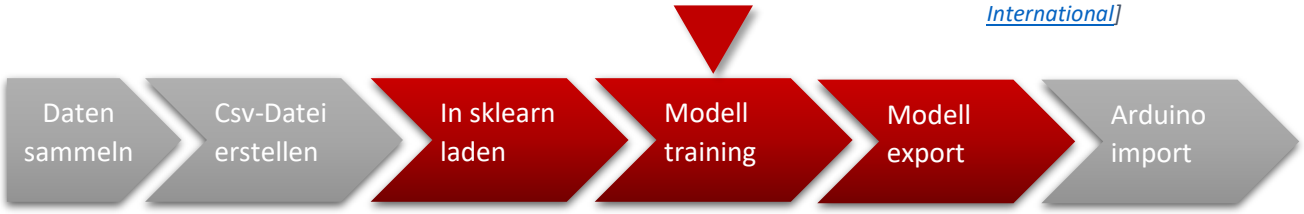


Abb. 23 Bild [Gemeinfrei] erzeugt mit DALL-E; Prompt „cute little python with a pencil doing complicated math, high quality illustration, digital art“ von Jörg [CC BY-SA 4.0 International]



Hinweis: Alle Skripte in diesem Dokument sind getestet (19.10.2022)

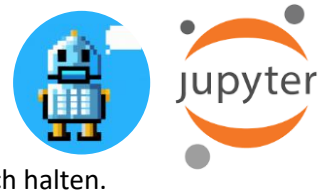
Alle folgenden Skripten – also Installationsroutinen, Pythonskripte in skLearn und Arduino-Quellcodes – sind getestet und validiert. Validiert bedeutet: die erstellten Modelle wurden auf den Arduino aufgespielt und getestet; die Inferenzen lagen für alle Modelle oberhalb der 90 Prozent. Die Skripten können also für den Einstieg im Unterricht eingesetzt werden, es sollte nicht für allzu viel Frust bei den Schülern sorgen.

Viele Tutorials, die man im Netz oder auch in Papers findet, funktionieren oftmals nur in ganz eingeschränkten Szenarien und lassen sich nicht mehr verhändern. Andere Quelltexte und Tutorials funktionieren oftmals gar nicht mehr, weil die Softwarepakete nicht weitergepflegt werden und deshalb Inkompatibiläten entstanden sind. Hier wäre es zwar theoretisch möglich, Parallel-Installationen von alten virtuellen Umgebungen für Python zu installieren, das ist aber für den praktischen Einsatz im Unterricht eigentlich nicht mögich. Deshalb sind solche Pakete wie „Everywherml“ oder „skPorter“ hier nicht verwendet worden.

Allgemein finden sich solche frustrierenden Rahmenbedingungen sehr oft im KI-Umfeld. Hier besteht eine zentrale Leistung des Dozenten/Lehrers m.M.n. darin, genau solche Dinge für den Unterricht zu vermeiden. Und das tut man, indem man die Materialien vorher ausprobiert.

> Besser, der Lehrer ist frustriert, als dass man stundenlang Unterrichtszeit verbrennt, wenn 15 oder mehr Schüler ihr Ziel nicht erreichen können.

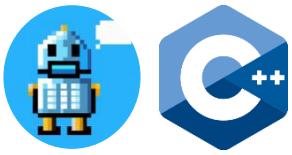
Eloquent: Decision-Tree-Klassifikation ohne Normalisierung



Der einfachste und trotzdem robuste ML-Algorithmus ist der Decision-Tree. Da es sich hierbei um eine Verzweigungskette von If-Verzweigungen handelt, bei denen die jeweiligen Werte vollständig unabhängig voneinander betrachtet werden, ist hier keine Normalisierung nötig. Deshalb kann man die sklearn-Skripte zunächst ebenfalls sehr einfach halten.

```
# DECISIONTREE CLASSIFICATION, RAW
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from micromlgen import port

df = pd.read_csv('LegoTraining.csv',sep=',')
df.head(3)
features = ['violet','blue','green','yellow','orange','red']
X = df[features]
y = df['label']
clf = DecisionTreeClassifier().fit(X, y)

predictions = clf.predict(X)
print("accuracy score: ", accuracy_score(y, predictions), "\n\n")

print(port(clf))
with open('decTreeClassifier.h', 'w') as file:
    file.write(port(clf))
```

Eloquent: Beispielhafter C++-Code für DecisionTree ohne Normalisierung



```
#pragma once

//#include <cstdarg>
```

Wichtig: Muss für Arduino auskommentiert werden

```
namespace Eloquent {
    namespace ML {
        namespace Port {
            class DecisionTree {
                public:
                    /**
                    * Predict class for features vector
                    */
                    int predict(float *x) {
                        if (x[0] <= 3157.5) {
                            if (x[3] <= 4839.0) {
                                if (x[1] <= 1162.5) {
                                    if (x[5] <= 1804.0) {
                                        return 0;
                                    }
                                    else {
                                        return 1;
                                    }
                                }
                                else {
                                    if (x[5] <= 3307.0) {
                                        return 3;
                                    }
                                    else {
                                        return 1;
```

... 😊

... hier wurde was weggelassen

Eloquent: Arduino-Code für den DecisionTree ohne Normalisierung


```
#include <Wire.h>
#include "Adafruit_AS726x.h"
#include "decTreeClassifier.h"

Eloquent::ML::Port::DecisionTree clf;

Adafruit_AS726x ams;
uint16_t mInt[AS726x_NUM_CHANNELS];
float m[AS726x_NUM_CHANNELS];

bool button = false;
void setup() {
  pinMode(2, INPUT);
  Serial.begin(9600);
  Wire.begin();
  if (!ams.begin()) {
    Serial.println("could not connect to sensor!");
    while (1);
  }
}

void loop() {
  button = digitalRead(2);
  if (button) {
    ams.drvOn();
    ams.startMeasurement();
    bool rdy = false;
    while (!rdy) {
      delay(5);
      rdy = ams.dataReady();
    }
    ams.readRawValues(mInt);
    for (int z = 0; z < 6; z++) {
      m[z] = (float)mInt[z];
    }
    float pred[6] = { m[0], m[1], m[2], m[3], m[4], m[5] };
    float label = clf.predict(pred);
    Serial.println(label);
  }
  ams.drvOff();
}
```

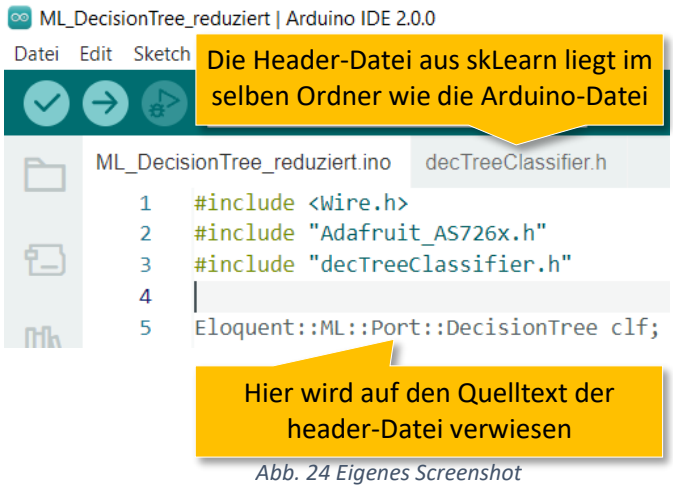
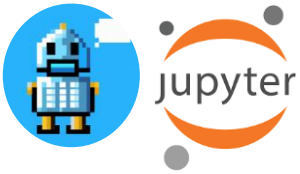


Abb. 24 Eigenes Screenshot

Eloquent: Decision-Tree-Klassifikation mit Normalisierung



```
# DECISIONTREE CLASSIFICATION, NORMALIZED
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from micromlgen import port

df = pd.read_csv('LegoTraining.csv',sep=',')
features = ['violet','blue','green','yellow','orange','red']

# C++ String für MinMaxScaler generieren:
cString = "float pred["+str(len(features))+"]={"
j=0;
for i in features:
    t1 = df.describe().loc['min',i]
    t2 = df.describe().loc['max',i]
    cString=cString+"(m["+str(j)+"]-"+str(t1)+")/("+str(t2)+"-"+str(t1)+"),"
    j=j+1
cString=cString[:-1]+"};"
print(cString)

X = df[features]
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
y = df['label']
clf = DecisionTreeClassifier().fit(X, y)

predictions = clf.predict(X)
print("accuracy score: ", accuracy_score(y, predictions), "\n\n")

print(port(clf))
with open('decTreeClassifier.h', 'w') as file:
    file.write(port(clf))
```

Eloquent: Beispielhafter C++-Code für DecisionTree mit Normalisierung

Achtung! Hier besteht der Quellcode nun aus zwei Teilen:

1. der Normalisierung
2. dem eigentlichen DecisionTree

Die Normalisierung wird in skLearn als MinMax-Variante gerechnet. Deshalb wird im Python-Code die entsprechende Codezeile für die Normalisierung mitgeneriert. Diese wird dann im Arduinocode eingesetzt, indem man das Array mit den Messwerten durch die folgende – hier exemplarisch dargestellte – Codezeile neu rechnet:

```
float pred[6] = { (m[0] - 197.0) / (16720.0 - 197.0), (m[1] - 229.0) / (11904.0 - 229.0), (m[2] - 315.0) / (49085.0 - 315.0), (m[3] - 354.0) / (51201.0 - 354.0), (m[4] - 365.0) / (51201.0 - 365.0), (m[5] - 236.0) / (19019.0 - 236.0) };
```

Ausschnitt aus dem normalisierten DecTree-Quellcode

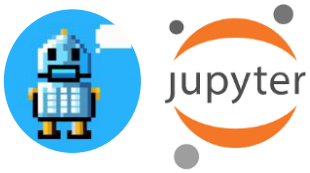
```
#pragma once

//#include <cstdarg>

namespace Eloquent {
    namespace ML {
        namespace Port {
            class DecisionTree {
                public:
                    /**
                    * Predict class for features vector
                    */
                    int predict(float *x) {
                        if (x[4] <= 0.39509403705596924) {
                            if (x[5] <= 0.20193792134523392) {
                                if (x[0] <= 0.013496338704499067) {
                                    return 0;
                                }

                                else {
                                    if (x[0] <= 0.2612721621990204) {
                                        if (x[2] <= 0.08831248804926872) {
                                            return 4;
                                        }

                                        else {
                                            return 3;
                                        }
                                    } ...
... 😊
... hier wurde was weggelassen
```

Eloquent: Arduino-Code für DecisionTree **mit** Normalisierung

```
#include <Wire.h>
#include "Adafruit_AS726x.h"
#include "decTreeClassifier.h"

Eloquent::ML::Port::DecisionTree clf;

Adafruit_AS726x ams;
uint16_t mInt[AS726x_NUM_CHANNELS];
float m[AS726x_NUM_CHANNELS];

bool button = false;
void setup() {
  pinMode(2, INPUT);
  Serial.begin(9600);
  Wire.begin();
  if (!ams.begin()) {
    Serial.println("could not connect to sensor.");
    while (1)
      ;
  }
}

void loop() {
  button = digitalRead(2);
  if (button) {
    ams.drvOn();
    ams.startMeasurement();
    bool rdy = false;
    while (!rdy) {
      delay(5);
      rdy = ams.dataReady();
    }
    ams.readRawValues(mInt);
    for (int z = 0; z < 6; z++) {
      m[z] = (float)mInt[z];
    }

    float pred[6] = { (m[0] - 197.0) / (16720.0 - 197.0), (m[1] - 229.0) / (11904.0 - 229.0), (m[2] - 315.0) / (49085.0 - 315.0), (m[3] - 354.0) / (51201.0 - 354.0), (m[4] - 365.0) / (51201.0 - 365.0), (m[5] - 236.0) / (19019.0 - 236.0) };

    float label = clf.predict(pred);
    Serial.println(label);
  }
  ams.drvOff();
}
```

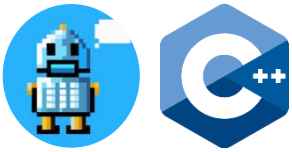
Veränderte Codezeile für die Normalisierungsberechnung

Eloquent: RandomForest-Klassifikation ohne Normalisierung



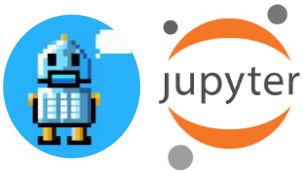
```
# RANDOM FOREST CLASSIFICATION, RAW
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from micromlgen import port

df = pd.read_csv('LegoTraining.csv',sep=',')
print(df.head(3))
features = ['violet','blue','green','yellow','orange','red']
X = df[features]
y = df['label']
clf = RandomForestClassifier(n_estimators=3).fit(X, y)

predictions = clf.predict(X)
print("accuracy score: ", accuracy_score(y, predictions), "\n\n")

print(port(clf))
with open('RFClassifier.h', 'w') as file:
    file.write(port(clf))
```

Eloquent: Beispielhafter C++-Code für Randomforest-Klassifikation mit 3 Bäumen ohne Normalisierung



```
#pragma once
//#include <cstdarg>
namespace Eloquent {
 namespace ML {
  namespace Port {
   class RandomForest {
    public:
     int predict(float *x) {
      uint8_t votes[5] = { 0 };

// tree #1
      if (x[4] <= 4358.5) {
       if (x[4] <= 1596.0) {
        if (x[0] <= 2118.5) {
         votes[3] += 1;
        }else {
         votes[2] += 1;
        }
       }else {
        if (x[3] <= 1920.0) {
         if (x[0] <= 1175.0) {
          if (x[5] <= 2240.0) {
           votes[0] += 1;
          }else {
           votes[1] += 1;
          }
         }else {
          votes[1] += 1;
         }
        }else {
         if (x[0] <= 3672.0) {
          votes[3] += 1;
         }else {
          votes[2] += 1;
         }
        }
       }
      }else {
       if (x[2] <= 3890.0) {
        votes[1] += 1;
       }else {
        votes[4] += 1;
       }
      }

// tree #2
      if (x[1] <= 1151.0) {
       if (x[3] <= 1434.0) {
        votes[1] += 1;
       }else {
        if (x[4] <= 2266.0) {
         votes[0] += 1;
        }else {
         votes[1] += 1;
        }
       }
      }else {
       if (x[3] <= 4808.5) {
        if (x[1] <= 2059.5) {
         if (x[3] <= 3436.5) {
          votes[3] += 1;
         }else {
          votes[1] += 1;
         }
        }else {
         if (x[2] <= 1981.5) {
          votes[2] += 1;
         }else {
          if (x[3] <= 1708.5) {
           votes[2] += 1;
          }else {
           if (x[3] <= 2506.0) {
            votes[3] += 1;
           }else {
            votes[2] += 1;
           }
          }
         }
        }
       }else {
        votes[4] += 1;
       }
      }

// tree #3
      if (x[3] <= 4839.0) {
       if (x[2] <= 1843.0) {
        if (x[0] <= 1175.0) {
         votes[0] += 1;
        }else {
         if (x[0] <= 3770.0) {
          votes[1] += 1;
         }else {
          votes[2] += 1;
         }
        }
       }else {
        if (x[5] <= 2924.0) {
         if (x[3] <= 1314.0) {
          votes[2] += 1;
         }else {
          if (x[4] <= 1792.5) {
           if (x[3] <= 1708.5) {
            if (x[4] <= 1280.5) {
             votes[3] += 1;
            }else {
             votes[2] += 1;
               … 😉
               … hier wurde
               … was weggelassen


      // return argmax of votes
      uint8_t classIdx = 0;
      float maxVotes = votes[0];
      for (uint8_t i = 1; i < 5; i++)
{
       if (votes[i] > maxVotes) {
        classIdx = i;
        maxVotes = votes[i];
       }
      }
      return classIdx;
     }
    protected:
    };
   }
  }
 }
```

Eloquent: XGBoost-Klassifikation mit 3 Bäumen ohne Normalisierung



Sofern noch nicht geschehen, muss die XGBoost-Library vorher installiert werden mit:

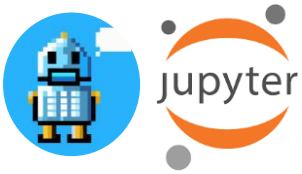
```
pip install xgboost
```

Der Quellcode sieht dann sehr ähnlich den bisherigen Quellcodes aus:

```
import pandas as pd
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from micromlgen import port

df = pd.read_csv('LegoTraining.csv',sep=',')
print(df.head(3))
features = ['violet','blue','green','yellow','orange','red']
X = df[features]
y = df['label']

clf = XGBClassifier(n_estimators=10)
clf.fit(X, y)

predictions = clf.predict(X)
print("accuracy score: ", accuracy_score(y, predictions), "\n\n")
print(port(clf, classname='XGBClassifier', tmp_file='xgboost.json'))

with open('XGBClassifier.h', 'w') as file:
    file.write(port(clf, classname='XGBClassifier', tmp_file='xgboost.json'))
```

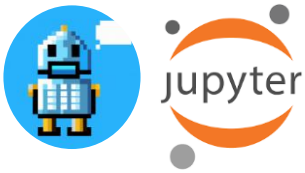
Auf eine Abbildung des C++-Quellcodes wird hier aufgrund des hohen Umfangs abgesehen, zudem der XGBoost dafür bekannt ist, dass aus dem Quellcode nichts mehr herausgelesen werden kann. Damit wird der didaktische Wert recht gering.

Eloquent: Naive-Bayes-Klassifikation mit Normalisierung



```
# NAIVE BAYES CLASSIFICATION, NORMALIZED
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.naive_bayes import GaussianNB
from micromlgen import port

def pick_best(X_train, X_test, y_train, y_test):
    best = (None, 0)
    for var_smoothing in range(-7, 1):
        clf = GaussianNB(var_smoothing=pow(10, var_smoothing))
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        accuracy = (y_pred == y_test).sum()
        if accuracy > best[1]:
            best = (clf, accuracy)
    print('best accuracy', best[1] / len(y_test), "\n\n")
    return best[0]

df = pd.read_csv('LegoTraining.csv',sep=',')
features = ['violet','blue','green','yellow','orange','red']

cString = "float pred["+str(len(features))+"]={"
j=0;
for i in features:
    t1 = df.describe().loc['min',i]
    t2 = df.describe().loc['max',i]
    cString=cString+"(x["+str(j)+"]-"+str(t1)+")/("+str(t2)+"-"+str(t1)+"),"
    j=j+1
cString=cString[:-1]+"};"
print(cString)

X = df[features]
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
clf = pick_best(X_train, X_test, y_train, y_test)

print(port(clf))
with open('naivbayes.h', 'w') as file:
    file.write(port(clf))
```

Eloquent: Naive-Bayes-Klassifikation ohne Normalisierung

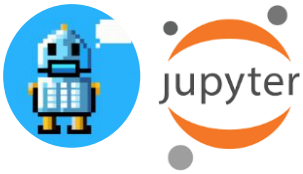


```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from micromlgen import port

def pick_best(X_train, X_test, y_train, y_test):
    best = (None, 0)
    for var_smoothing in range(-7, 1):
        clf = GaussianNB(var_smoothing=pow(10, var_smoothing))
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        accuracy = (y_pred == y_test).sum()
        if accuracy > best[1]:
            best = (clf, accuracy)
    print('best accuracy', best[1] / len(y_test), "\n\n")
    return best[0]

df = pd.read_csv('LegoTraining.csv',sep=',')
print(df.head(3))
features = ['violet','blue','green','yellow','orange','red']

X = df[features]
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
clf = pick_best(X_train, X_test, y_train, y_test)

print(port(clf))
with open('naivbayes.h', 'w') as file:
    file.write(port(clf))
```

Eloquent: Beispielhafter C++-Code für Naive-Bayes mit Normalisierung

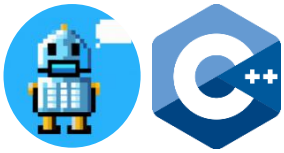
```
float pred[6]={
(x[0]- 518.0)/(9954.0 -518.0),
(x[1]- 573.0)/(5382.0 -573.0),
(x[2]-1043.0)/(13119.0-1043.0),
(x[3]-1084.0)/(15600.0-1084.0),
(x[4]- 771.0)/(19199.0-771.0),
(x[5]- 348.0)/(8464.0 -348.0)};

#pragma once
//#include <cstdarg>
namespace Eloquent {
 namespace ML {
  namespace Port {
   class GaussianNB {
    public:
     /**
     * Predict class for features vector
     */
     int predict(float *x) {
      float votes[5] = { 0.0f };
      float th[6] = { 0 };
      float sig[6] = { 0 };
      th[0] = 0.067; th[1] = 0.076; th[2] = 0.058; th[3] = 0.033; th[4] = 0.070; th[5] = 0.113;
      sig[0] = 1e-08; sig[1] = 2e-08; sig[2] = 9e-09; sig[3] = 9e-09; sig[4] = 1e-08; sig[5] = 1e-08;
      votes[0] = 0.179 - gauss(x, th, sig);

      th[0] = 0.109; th[1] = 0.055; th[2] = 0.058; th[3] = 0.048; th[4] = 0.103; th[5] = 0.327;
      sig[0] = 0.003; sig[1] = 0.005; sig[2] = 0.004; sig[3] = 0.003; sig[4] = 0.004; sig[5] = 0.005;
      votes[1] = 0.179 - gauss(x, th, sig);

      th[0] = 0.573; th[1] = 0.475; th[2] = 0.166; th[3] = 0.069; th[4] = 0.060; th[5] = 0.038;
      sig[0] = 0.039; sig[1] = 0.034; sig[2] = 0.004; sig[3] = 0.006; sig[4] = 0.002; sig[5] = 0.001;
      votes[2] = 0.230 - gauss(x, th, sig);

      th[0] = 0.050; th[1] = 0.295; th[2] = 0.157; th[3] = 0.046; th[4] = 0.042; th[5] = 0.027;
      sig[0] = 0.002; sig[1] = 0.008; sig[2] = 0.008; sig[3] = 0.001; sig[4] = 0.002; sig[5] = 0.001;
      votes[3] = 0.153 - gauss(x, th, sig);

      th[0] = 0.087; th[1] = 0.299; th[2] = 0.653; th[3] = 0.648; th[4] = 0.604; th[5] = 0.631;
      sig[0] = 0.003; sig[1] = 0.009; sig[2] = 0.043; sig[3] = 0.034; sig[4] = 0.044; sig[5] = 0.029;
      votes[4] = 0.256 - gauss(x, th, sig);

      // return argmax of votes
      uint8_t classIdx = 0;
      float maxVotes = votes[0];
      for (uint8_t i = 1; i < 5; i++) {
       if (votes[i] > maxVotes) {
        classIdx = i;
        maxVotes = votes[i];
       }
      }
      return classIdx;
     }
    protected:
     /**
     * Compute gaussian value
     */
     float gauss(float *x, float *th, float *sig) {
      float gauss = 0.0f;
      for (uint16_t i = 0; i < 6; i++) {
       gauss += log(sig[i]);
       gauss += abs(x[i] - th[i]) / sig[i];
      }
      return gauss;
     }
    };
   }
  }
 }
```

Eloquent: SupportVectorMachine-Klassifikation ohne Normalisierung



```
import pandas as pd
from sklearn.svm import SVC
from micromlgen import port
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score

df = pd.read_csv('LegoTraining.csv',sep=',')
print(df.head(3))
features = ['violet','blue','green','yellow','orange','red']

cString = "float pred["+str(len(features))+"]={"
j=0;
for i in features:
    t1 = df.describe().loc['min',i]
    t2 = df.describe().loc['max',i]
    cString=cString+"(m["+str(j)+"]-"+str(t1)+")/("+str(t2)+"-"+str(t1)+"),"
    j=j+1
cString=cString[:-1]+"};"
print(cString)

X = df[features]
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

y = df['label']
clf = SVC(kernel='linear',gamma=0.001).fit(X,y)

predictions = clf.predict(X)
print("accuracy score: ", accuracy_score(y, predictions), "\n\n")

print(port(clf))
with open('linearSVM_norm.h', 'w') as file:
    file.write(port(clf))
```

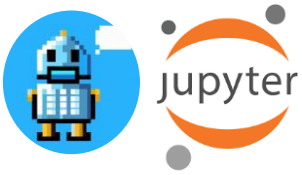
Eloquent: Beispielhafter C++-Code für SVM ohne Normalisierung



```
#pragma once
//#include <cstdarg>
namespace Eloquent {
 namespace ML {
  namespace Port {
   class SVM {
    public:
     int predict(float *x) {
      float kernels[20] = { 0 };
      float decisions[10] = { 0 };
      int votes[5] = { 0 };
      kernels[0] =compute_kernel(x,1155.0,941.0,1752.0,1574.0,2068.0,1265.0);
      kernels[1] =compute_kernel(x,1156.0,943.0,1753.0,1576.0,2070.0,1266.0);
      kernels[2] =compute_kernel(x,1157.0,941.0,1753.0,1574.0,2069.0,1264.0);
      kernels[3] =compute_kernel(x,1156.0,943.0,1753.0,1575.0,2071.0,1265.0);
      kernels[4] =compute_kernel(x,2813.0,1689.0,3606.0,3894.0,5073.0,3980.0);
      kernels[5] =compute_kernel(x,2178.0,914.0,1447.0,1719.0,2025.0,2405.0);
      kernels[6] =compute_kernel(x,1349.0,663.0,1644.0,1892.0,1851.0,2565.0);
      kernels[7] =compute_kernel(x,1193.0,573.0,1326.0,1223.0,1703.0,2342.0);
      kernels[8] =compute_kernel(x,3513.0,1968.0,2001.0,1216.0,1307.0,525.0);
      kernels[9] =compute_kernel(x,3502.0,1741.0,1330.0,1500.0,771.0,580.0);
      kernels[10]=compute_kernel(x,3809.0,1915.0,4270.0,2974.0,3050.0,726.0);
      kernels[11]=compute_kernel(x,1206.0,2229.0,3356.0,4272.0,3322.0,2634.0);
      kernels[12]=compute_kernel(x,735.0,2518.0,1500.0,1248.0,966.0,854.0);
      kernels[13]=compute_kernel(x,1451.0,1815.0,4783.0,1955.0,3249.0,434.0);
      kernels[14]=compute_kernel(x,2558.0,5382.0,2274.0,2344.0,1100.0,2405.0);
      kernels[15]=compute_kernel(x,690.0,1704.0,1933.0,1412.0,970.0,450.0);
      kernels[16]=compute_kernel(x,900.0,1382.0,2556.0,1948.0,1603.0,476.0);
      kernels[17]=compute_kernel(x,1664.0,2611.0,2620.0,2185.0,1881.0,793.0);
      kernels[18]=compute_kernel(x,1901.0,3471.0,6188.0,7418.0,4757.0,2740.0);
      kernels[19]=compute_kernel(x,1588.0,1578.0,4174.0,5406.0,5902.0,3111.0);
      decisions[0] = 1.19+kernels[0]*1.23e-06+kernels[4]*-4.50e-08+kernels[5]*-1.59e-07+kernels[7]*-1.02e-06;
      decisions[1] = 1.47+kernels[2]*2.60e-07+kernels[8]*-6.60e-08+kernels[9]*-1.487e-07+kernels[10]*
                     -4.57e-08;
      decisions[2] = 1.44+kernels[1]*1.170e-06+kernels[11]*-1.40598e-07+kernels[12]*-4.587e-08+kernels[15]*
                     -4.18e-07+kernels[16]*-5.65e-07;
      decisions[3] = 2.10+kernels[3]*5.10e-08+kernels[19]*-5.10e-08;
      decisions[4] = 0.61+kernels[5]*2.91e-07+kernels[9]*-2.36e-07+kernels[10]*-5.93e-08;
      decisions[5] = 0.32+kernels[4]*1.57e-07+kernels[6]*3.32e-07+kernels[11]*-3.28e-07+kernels[12]*
                     -5.35e-09+kernels[15]*-1.58e-07;
      decisions[6] = 2.81+kernels[4] *3.59e-07+ kernels[19]*-3.596e-07;
      decisions[7] =-1.46+kernels[8] *2.76-07+kernels[10]*1.80e-07+kernels[13]*-6.23e-08+kernels[14]*
                     -1.64e-09+kernels[17]*-3.93e-07;
      decisions[8] = 1.73+kernels[10]*8.08e-08+kernels[18]*-4.78e-09+kernels[19]*-7.60e-08;
      decisions[9] = 4.59+kernels[11]*2.12e-07+kernels[18]*-5.05e-09+kernels[19]*-2.07e-07;
      votes[decisions[0] > 0 ? 0 : 1] += 1;
      votes[decisions[1] > 0 ? 0 : 2] += 1;
      votes[decisions[2] > 0 ? 0 : 3] += 1;
      votes[decisions[3] > 0 ? 0 : 4] += 1;
      votes[decisions[4] > 0 ? 1 : 2] += 1;
      votes[decisions[5] > 0 ? 1 : 3] += 1;
      votes[decisions[6] > 0 ? 1 : 4] += 1;
      votes[decisions[7] > 0 ? 2 : 3] += 1;
      votes[decisions[8] > 0 ? 2 : 4] += 1;
      votes[decisions[9] > 0 ? 3 : 4] += 1;
      int val = votes[0];
      int idx = 0;
      for (int i = 1; i < 5; i++) {
       if (votes[i] > val) {
        val = votes[i];
        idx = i;
       }
      }
      return idx;
     }
    protected:
     /* Compute kernel between feature vector and support vector.
      * Kernel type: linear*/
     float compute_kernel(float *x, ...) {
      va_list w;
      va_start(w, 6);
      float kernel = 0.0;
      for (uint16_t i = 0; i < 6; i++) {
       kernel += x[i] * va_arg(w, double);
      }
      return kernel;
     }
    };
   }
  }
 }
```

Eloquent: Regressions-Modelle

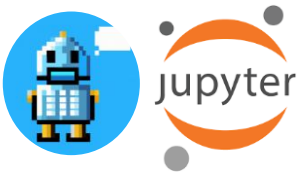
Der Vollständigkeit halber sollen hier nicht nur die Klassifikations- sondern auch die Regressionsmodelle wiedergegeben werden. Hierbei wird eine kontinuierliche Vorhersage angestrebt.




```
# DECISIONTREE REGRESSION, RAW
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn import metrics
from micromlgen import port

df = pd.read_csv('LegoTraining.csv',sep=',')
#print(df.head(5))
features = ['violet','blue','green','yellow','orange','red']
X = df[features]
y = df['label']
regr = DecisionTreeRegressor(max_depth= 7, min_samples_leaf=2).fit(X, y)

predictions = regr.predict(X)
print('MAE: ', metrics.mean_absolute_error(y, predictions))
print('MSE: ', metrics.mean_squared_error(y, predictions))


print(port(regr))
with open('C:\Test\decTreeRegressor.h', 'w') as file:
    file.write(port(regr))
```

```
# RANDOMFOREST REGRESSION, RAW
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from micromlgen import port

df = pd.read_csv('LegoTraining.csv',sep=',')
print(df.head(5))
features = ['violet','blue','green','yellow','orange','red']
X = df[features]
y = df['label']
regr = RandomForestRegressor(n_estimators=3).fit(X, y)

predictions = regr.predict(X)
print('MAE: ', metrics.mean_absolute_error(y, predictions))
print('MSE: ', metrics.mean_squared_error(y, predictions))

print(port(regr))
with open('RForestRegressor.h', 'w') as file:
    file.write(port(regr))
```

Eloquent: Principal Component Analysis, PCA



Bei der PCA handelt es sich um eine Dimensionsreduktion, die auch als Datenkomprimierung interpretiert werden kann. Bei vielen gleichzeitig aufgenommenen Messwerten kann so eine Reduzierung der Datenmenge erreicht werden.

Eloquent: Der Python-Code der PCA, zusammen mit einem DecisionTree ohne Normalisierung:

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from micromlgen import port

df = pd.read_csv('LegoTraining.csv',sep=',')
print(df.head(3))
features = ['violet','blue','green','yellow','orange','red']
X = df[features]
y = df['label']

# PCA von 6 auf 3 Dimensionen ----------------------------------
pca = PCA(n_components=3)
principalComponents = pca.fit(X)
print(port(principalComponents))

# ursprüngliche Werte transformieren, damit das Modell trainiert werden kann
pcaValues = pca.transform(X)
# Neuen Dataframe zusammmensetzen
principalDf = pd.DataFrame(data = pcaValues, columns = ['pc1', 'pc2', 'pc3'])
principalDf['label']=y

# Training analog zu bisherigem Prozedere am Beispiel Decisiontree
featuresPCA = ['pc1', 'pc2', 'pc3']
Xp = principalDf[featuresPCA]
yp = principalDf['label']
clf = DecisionTreeClassifier().fit(Xp, yp)

predictions = clf.predict(Xp)
print("accuracy score: ", accuracy_score(yp, predictions), "\n\n")

print(port(clf))
with open('decTreeClassifier.h', 'w') as file:
    file.write(port(clf))
```

Eloquent: Beispielhafter C++-Quellcode für die PCA ohne Normalisierung:

```
#pragma once
//#include <cstdarg>
namespace Eloquent {
    namespace ML {
        namespace Port {
            class PCA {
                public:
                    /**
                    * Apply dimensionality reduction
                    * @warn Will override the source vector if no dest provided!
                    */
                    void transform(float *x, float *dest = NULL) {
                        static float u[3] = { 0 };
                        u[0] = dot(x,-0.055,0.029,0.436,0.556,0.652,0.266);
                        u[1] = dot(x,0.889,0.349,0.190,0.087,-0.063,-0.194);
                        u[2] = dot(x,-0.365,0.206,0.471,0.350,-0.434,-0.538);
                        memcpy(dest != NULL ? dest : x, u, sizeof(float) * 3);
                    }
                protected:
                    /**
                    * Compute dot product with varargs
                    */
                    float dot(float *x, ...) {
                        va_list w;
                        va_start(w, 6);
                        static float mean[] = {2273,1813,3648,3551,3865,2140};
                        float dot = 0.0;
                        for (uint16_t i = 0; i < 6; i++) {
                            dot += (x[i] - mean[i]) * va_arg(w, double);
                        }
                        return dot;
                    }
                };
    }}}
```

Eloquent: Zur PCA zugehöriger beispielhafter DecisionTree-C++ Code:

```
#pragma once
//#include <cstdarg>
namespace Eloquent {
    namespace ML {
        namespace Port {
            class DecisionTree {
                public:
                    /**
                    * Predict class for features vector
                    */
                    int predict(float *x) {
                        if (x[0] <= 2147.8147583007812) {
                            if (x[1] <= 485.1079730987549) {
                                if (x[2] <= -444.431339263916) {
                                    return 1;
                                }
                                else {
                                    if (x[2] <= 226.7122344970703) {
                                        return 0;
                                    }
                                    else {
                                        return 3;
                                    }
                                }
                            }
                            else {
                                if (x[2] <= 601.1160278320312) {
                                    return 2;
                                }
                                else {
                                    if (x[1] <= 1526.6473388671875) {
                                        return 3;
                                    }
                                    else {
                                        return 2;
                                    }
                                }
                            }
                        }
                        else {
                            return 4;
                        }
                    }
                protected: };} } }
```

Eloquent: Arduino-Quellcode, der PCA und Decision-Tree verbindet

```
#include <Wire.h>
#include "Adafruit_AS726x.h"
#include "decTreeClassifier.h"
#include "pca.h"

Eloquent::ML::Port::DecisionTree clf;
Eloquent::ML::Port::PCA pca;

Adafruit_AS726x ams;
uint16_t mInt[AS726x_NUM_CHANNELS];
float m[AS726x_NUM_CHANNELS];

bool button = false;
void setup() {
  pinMode(2, INPUT);
  Serial.begin(9600);
  Wire.begin();
  if (!ams.begin()) {
    Serial.println("could not connect to sensor!");
    while (1);
  }
}

void loop() {
  button = digitalRead(2);
  if (button) {
    ams.drvOn();
    ams.startMeasurement();
    bool rdy = false;
    while (!rdy) {
      delay(5);
      rdy = ams.dataReady();
    }
    ams.readRawValues(mInt);
    for (int z = 0; z < 6; z++) {
      m[z] = (float)mInt[z];
    }

    float pca_input[6] = { m[0], m[1], m[2], m[3], m[4], m[5] };
    float pca_output[3];
    pca.transform(pca_input, pca_output);

    float label = clf.predict(pca_output);
    Serial.println(label);
  }
  ams.drvOff();
}
```

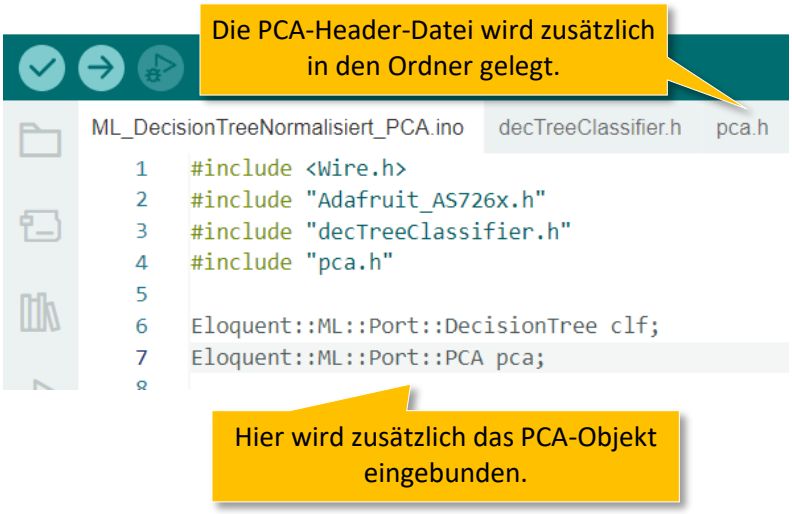
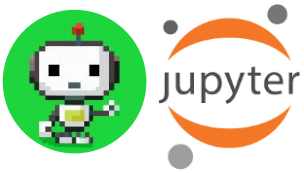


Abb. 25 Eigenes Screenshot

Zusätzlicher Code für die PCA: exakt wie in skLearn-Pythonskript angegeben, wird ein Messwertarray an die PCA übergeben, und diese rechnet sie in ein neues Messwertarray mit niedrigerer Dimension um.

EM-Learn: Decision-Tree-Klassifikation ohne Normalisierung



https://github.com/emlearn/emlearn/tree/master/emlearn

folgende Bibliothek muss später in den Arduino-Ordner eingebunden werden:
https://github.com/emlearn/emlearn/blob/master/emlearn/eml_trees.h

```
# DECISION TREE CLASSIFICATION, RAW
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import os.path
import emlearn

df = pd.read_csv('neueAufnahmen.csv',sep=',')
features = ['violet','blue','green','yellow','orange','red']
X = df[features]
y = df['label']
clf = DecisionTreeClassifier().fit(X, y)

predictions = clf.predict(X)
print("accuracy score: ", accuracy_score(y, predictions), "\n\n")

cmodel = emlearn.convert(clf, method='inline')
code = cmodel.save(file='DecisionTree.h', name='DTC')
print(code)
```

EM-Learn: Beispielhafter C++-Quellcode DecisionTree ohne Normalisierung:

```
#include "eml_trees.h"
EmlTreesNode DTC_nodes[16] = {
  { 4, 20450.0, 1, 15 },
  { 5, 4029.0, 2, 12 },
  { 1, 926.0, 3, 4 },
  { -1, 0, -1, -1 },
  { 0, 4514.0, 5, 8 },
  { 1, 2830.5, 6, 7 },
  { -1, 4, -1, -1 },
  { -1, 3, -1, -1 },
  { 3, 3515.0, 9, 10 },
  { -1, 5, -1, -1 },
  { 5, 2249.0, 6, 11 },
  { 0, 13063.0, 7, 9 },
  { 1, 596.0, 13, 14 },
  { -1, 1, -1, -1 },
  { -1, 6, -1, -1 },
  { -1, 2, -1, -1 }
};
int32_t DTC_tree_roots[1] = { 0 };
EmlTrees DTC = {
        16,
        DTC_nodes,
        1,
        DTC_tree_roots,
    };
static inline int32_t DTC_predict_tree_0(const float *features, int32_t features_length) {
          if (features[4] < 20450.0) {
              if (features[5] < 4029.0) {
                  if (features[1] < 926.0) {
                      return 0;
                  } else {
                      if (features[0] < 4514.0) {
                          if (features[1] < 2830.5) {
                              return 4;
                          } else {
                              return 3;
```

… 😊

… hier wurde

… was weggelassen

EM-Learn: Arduino-Code für den DecisionTree ohne Normalisierung

https://github.com/emlearn/emlearn/tree/master/emlearn

folgende Bibliothek muss später in den Arduino-Ordner eingebunden werden:
https://github.com/emlearn/emlearn/blob/master/emlearn/eml_trees.h

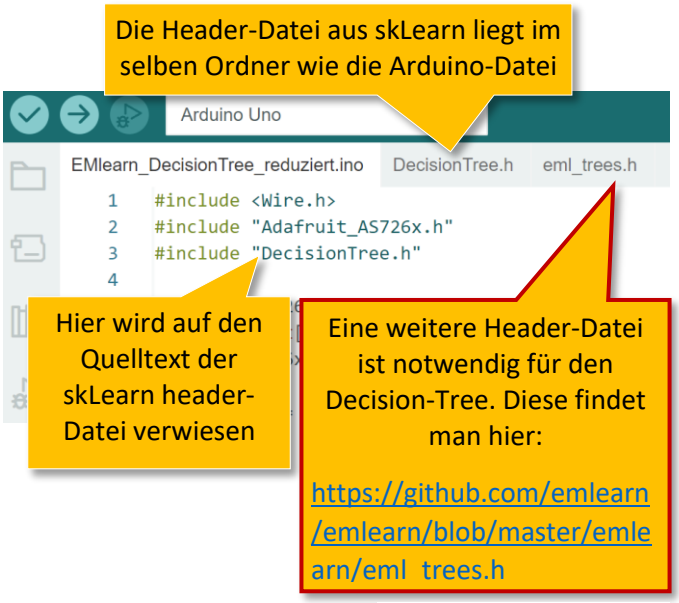


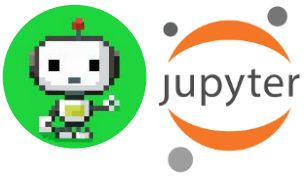
Abb. 26 Eigenes Screenshot

```
#include <Wire.h>
#include "Adafruit_AS726x.h"
#include "DecisionTree.h"

Adafruit_AS726x ams;
uint16_t mInt[AS726x_NUM_CHANNELS];
float m[AS726x_NUM_CHANNELS];

bool button = false;
void setup() {
  pinMode(2, INPUT);
  Serial.begin(9600);
  Wire.begin();
  if (!ams.begin()) {
    Serial.println("could not connect to sensor!");
    while (1);
  }
}
void loop() {
  button = digitalRead(2);
  if (button) {
    ams.drvOn();
    ams.startMeasurement();
    bool rdy = false;
    while (!rdy) {
      delay(5);
      rdy = ams.dataReady();
    }
    ams.readRawValues(mInt);
    for (int z = 0; z < 6; z++) {
      m[z] = (float)mInt[z];
    }
    float pred[6] = { m[0], m[1], m[2], m[3], m[4], m[5] };

    int label = DTC_predict_tree_0(pred, 6);

    Serial.println(label);
  }
  ams.drvOff();
}
```

Aufruf des DecisionTrees aus dem Quelltext der Header-Datei

EM-Learn: Random-Forest-Klassifikation ohne Normalisierung

https://github.com/emlearn/emlearn/tree/master/emlearn



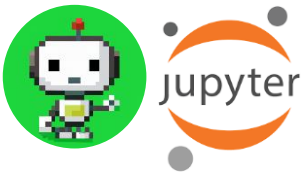
folgende Bibliothek muss später in den Arduino-Ordner eingebunden werden:
https://github.com/emlearn/emlearn/blob/master/emlearn/eml_trees.h

```
# RANDOM FOREST CLASSIFICATION, RAW
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import os.path
import emlearn

df = pd.read_csv('neueAufnahmen.csv',sep=',')
print(df.head(3))
features = ['violet','blue','green','yellow','orange','red']
X = df[features]
y = df['label']
clf = RandomForestClassifier(n_estimators=3).fit(X, y)

predictions = clf.predict(X)
print("accuracy score: ", accuracy_score(y, predictions), "\n\n")

cmodel = emlearn.convert(clf, method='inline')
code = cmodel.save(file='EML_RandForest.h', name='RFO')
print(code)
```

Der Arduinocode ist derselbe, deshalb wird er hier nicht erneut abgebildet.

EM-Learn: NeuralNet-Klassifikation mit Normalisierung

https://github.com/emlearn/emlearn/tree/master/emlearn



folgende ZWEI Bibliotheken müssen später in den Arduino-Ordner eingebunden werden:
https://github.com/emlearn/emlearn/blob/master/emlearn/eml_net.h
https://github.com/emlearn/emlearn/blob/master/emlearn/eml_common.h

```
# NEURAL NET CLASSIFICATION, NORMALIZED
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
import os.path
import emlearn

df = pd.read_csv('neueAufnahmen.csv',sep=',')
features = ['violet','blue','green','yellow','orange','red']

cString = "float pred["+str(len(features))+"]={"
j=0;
for i in features:
    t1 = df.describe().loc['min',i]
    t2 = df.describe().loc['max',i]
    cString=cString+"(m["+str(j)+"]-"+str(t1)+")/("+str(t2)+"-"+str(t1)+"),"
    j=j+1
cString=cString[:-1]+"};"
print(cString)

X = df[features]
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

y = df['label']

nnclf = MLPClassifier( hidden_layer_sizes=(8,7), max_iter=3000, random_state=1 ).fit(X,
y)

predictions = nnclf.predict(X)
print("accuracy score: ", accuracy_score(y, predictions), "\n\n")

cmodel = emlearn.convert(nnclf)
code = cmodel.save(file='EML_NeuralNet.h', name='NN')
print(code)
```

Hyperparameter für den Aufbau des Neuronalen Netzes

Erläuterung der Netztopologie:

Bei den hidden_layer_sizes wird durch Kommata getrennt die Netztopologie bestimmt. Da der verwendete Sensor insgesamt 6 Messwerte generiert, die auch alle als Input genutzt werden, folgt:

- 6 Inputneuronen werden automatisch generiert.
- Ebenfalls die hier verwendeten 8 verschiedenen Output-Klassen werden automatisch gesetzt – falls auch tatsächlich alle 8 Klassen verwendet wurden. In dem hier verwendeten Beispieldatensatz werden 5 Output-klassen verwendet: (siehe Seite 13: 0: Nichts, 1: Rot, 2: Orange, 3: Grün, 4: Türkis)

Rot: 1	Orange: 2	Gelb: 2
Grün: 3	Türkis: 4	Blau: 5
Violett: 6	Nichts: 0	

Deshalb sind die beiden Layer Input: 6 Neuronen und Output: 5 Neuronen automatische gesetzt.

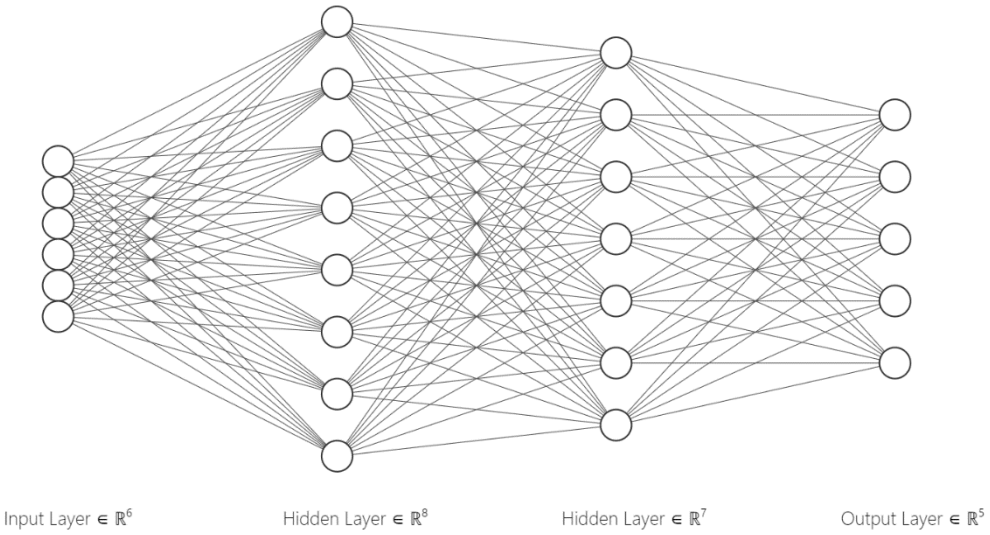


Abb. 27 Netztopologie aus dem obigen Quelltext: links 6 Inputlayer, rechts 5 Outputlayer

Normalisierung

Künstliche Neuronen benötigen für den Lernprozess solche Zahlenwert, die möglichst zwischen den Werte -1 und + 1 liegen sollten.

Deshalb ist die Normalisierung der Daten zwingend notwendig. Es wird wieder der MinMax-Scaler verwendet, der sich sowohl im Jupyter-Pythonskript als auch im Arduino-Quellcode wiederfindet.

EM-Learn: Arduino-Code für das Neuronale Netz mit Normalisierung

Aus dem skLearn-Skript wird aus dem Output wieder die folgende Quelltextzeile für die MinMax-Normalsierung per Copy-und-Paste übernommen:

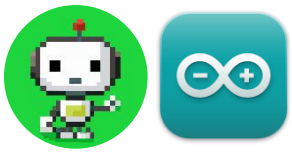
```
float pred[6]={(m[0]-158.0)/(39053.0-158.0),(m[1]-178.0)/(17213.0-178.0),(m[2]-
365.0)/(51201.0-365.0),(m[3]-419.0)/(51201.0-419.0),(m[4]-472.0)/(51201.0-472.0),(m[5]-
235.0)/(20243.0-235.0)};
```

… und an der entsprechenden Stelle im Arduino-Code eingebaut (siehe nächste Seite).

Ausserdem müssen hier zwei Header-Dateien zusätzlich hinzukopiert werden:

Abb. 28 Eigenes Screenshot

ACHTUNG: Da die Header-Dateien nicht im regulären Library-Verzeichnis, sondern im Quelltext-Ordner liegen, muss die folgende Zeile korrigiert werden:



```
#include <Wire.h>
#include "Adafruit_AS726x.h"
#include "EML_NeuralNet.h"

Adafruit_AS726x ams;
uint16_t mInt[AS726x_NUM_CHANNELS];
float m[AS726x_NUM_CHANNELS];

bool button = false;
void setup() {
  pinMode(2, INPUT);
  Serial.begin(9600);
  Wire.begin();
  if (!ams.begin()) {
    Serial.println("could not connect to sensor!");
    while (1);
  }
}
void loop() {
  button = digitalRead(2);
  if (button) {
    ams.drvOn();
    ams.startMeasurement();
    bool rdy = false;
    while (!rdy) {
      delay(5);
      rdy = ams.dataReady();
    }
    ams.readRawValues(mInt);
    for (int z = 0; z < 6; z++) {
      m[z] = (float)mInt[z];
    }
```

MinMax-Normalisierung

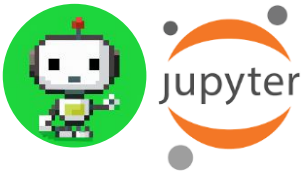
```
    float pred[6]={(m[0]-197.0)/(16720.0-197.0),(m[1]-229.0)/(11904.0-229.0),(m[2]-315.0)/(49085.0-315.0),(m[3]-354.0)/(51201.0-354.0),(m[4]-365.0)/(51201.0-365.0),(m[5]-236.0)/(19019.0-236.0)};
```

```
    int label = NN_predict(pred, 6);
```

Aufruf des Neuronalen Netzes aus dem Quelltext der Header-Datei

```
    Serial.println(label);
  }
  ams.drvOff();
}
```

EM-Learn: NaiveBayes-Klassifikation mit Normalisierung



https://github.com/emlearn/emlearn/tree/master/emlearn

folgende DREI Bibliotheken müssen später in den Arduino-Ordner eingebunden werden:
https://github.com/emlearn/emlearn/blob/master/emlearn/eml_bayes.h
https://github.com/emlearn/emlearn/blob/master/emlearn/eml_common.h
https://github.com/emlearn/emlearn/blob/master/emlearn/eml_fixedpoint.h

```
# NAIVE BAYES CLASSIFICATION, NORMALIZED
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.naive_bayes import GaussianNB
import os.path
import emlearn

def pick_best(X_train, X_test, y_train, y_test):
    best = (None, 0)
    for var_smoothing in range(-7, 1):
        clf = GaussianNB(var_smoothing=pow(10, var_smoothing))
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        accuracy = (y_pred == y_test).sum()
        if accuracy > best[1]:
            best = (clf, accuracy)
    print('best accuracy', best[1] / len(y_test), "\n\n")
    return best[0]

df = pd.read_csv('neueAufnahmen.csv',sep=',')
features = ['violet','blue','green','yellow','orange','red']

cString = "float pred["+str(len(features))+"]={"
j=0;
for i in features:
    t1 = df.describe().loc['min',i]
    t2 = df.describe().loc['max',i]
    cString=cString+"(x["+str(j)+"]-"+str(t1)+")/("+str(t2)+"-"+str(t1)+"),"
    j=j+1
cString=cString[:-1]+"};"
print(cString)

X = df[features]
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
clf = pick_best(X_train, X_test, y_train, y_test)

cmodel = emlearn.convert(clf)
code = cmodel.save(file='EML_NaiveBayes.h', name='NAB')
print(code)
```

EM-Learn: Arduino-Code für das Neuronale Netz mit Normalisierung

Aus dem skLearn-Skript wird aus dem Output wieder die folgende Quelltextzeile für die MinMax-Normalsierung per Copy-und-Paste übernommen:

```
float pred[6] = { (m[0] - 158.0) / (39053.0 - 158.0), (m[1] - 178.0) / (17213.0 - 178.0), (m[2] - 365.0) / (51201.0 - 365.0), (m[3] - 419.0) / (51201.0 - 419.0), (m[4] - 472.0) / (51201.0 - 472.0), (m[5] - 235.0) / (20243.0 - 235.0) };
```

… und an der entsprechenden Stelle im Arduino-Code eingebaut (siehe nächste Seite).

Ausserdem müssen hier drei Header-Dateien zusätzlich hinzukopiert werden:

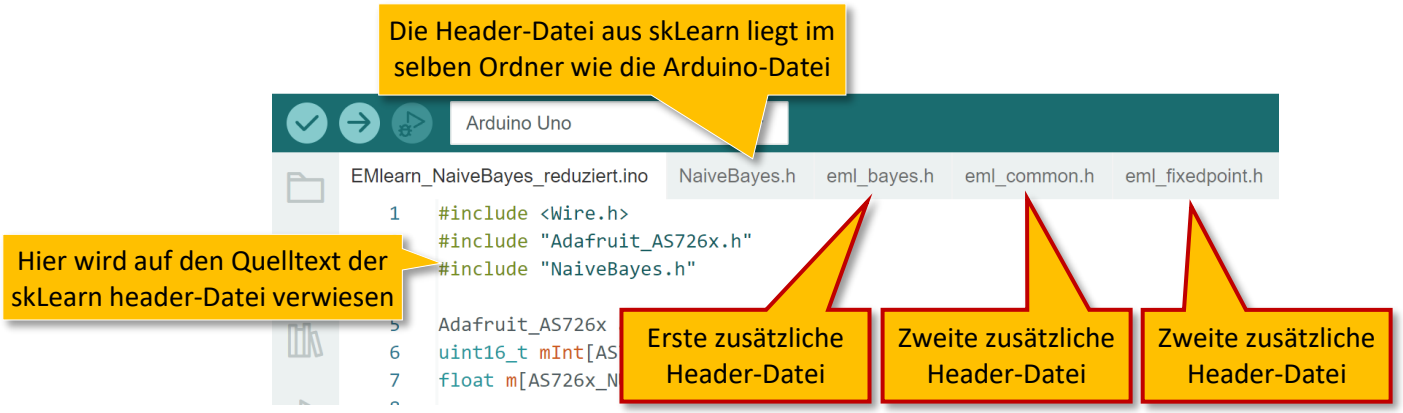
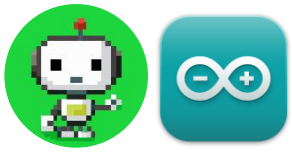


Abb. 29 Eigenes Screenshot

ACHTUNG: Da die Header-Dateien nicht im regulären Library-Verzeichnis, sondern im Quelltext-Ordner liegen, muss die folgende Zeile korrigiert werden:

~~`#include <eml_bayes.h>`~~ → `#include "eml_bayes.h"`



```
#include <Wire.h>
#include "Adafruit_AS726x.h"
#include "NaiveBayes.h"

Adafruit_AS726x ams;
uint16_t mInt[AS726x_NUM_CHANNELS];
float m[AS726x_NUM_CHANNELS];

bool button = false;
void setup() {
  pinMode(2, INPUT);
  Serial.begin(9600);
  Wire.begin();
  if (!ams.begin()) {
    Serial.println("could not connect to sensor!");
    while (1)
      ;
  }
}

void loop() {
  button = digitalRead(2);
  if (button) {
    ams.drvOn();
    ams.startMeasurement();
    bool rdy = false;
    while (!rdy) {
      delay(5);
      rdy = ams.dataReady();
    }
    ams.readRawValues(mInt);
    for (int z = 0; z < 6; z++) {
      m[z] = (float)mInt[z];
    }
```

MinMax-Normalisierung

```
    float pred[6] = { (m[0] - 158.0) / (39053.0 - 158.0), (m[1] - 178.0) / (17213.0 - 178.0), (m[2] - 365.0) / (51201.0 - 365.0), (m[3] - 419.0) / (51201.0 - 419.0), (m[4] - 472.0) / (51201.0 - 472.0), (m[5] - 235.0) / (20243.0 - 235.0) };

    int label = NAB_predict(pred, 6);
    Serial.println(label);
  }
  ams.drvOff();
}
```

Aufruf des NaiveBayes-Klassifizierers aus dem Quelltext der Header-Datei