Hardware & Software Codesign Academy
ISPU challenge

STMicroelectronics

MEMS subGroup – MEMS Software Solutions team

Group 1

Index

1. Introduction
2. Dataset Preprocessing and Input train set creation
3. Decision tree VS KNN
4. Feature importance method for parameter selection (grid search, skyline algorithm)
5. Convert from Python a simil C with tree_to_code function
6. Importing on main.c (/algo_00) features computation (Acc_nomr, Gyro_norm, Var_Acc_x) usando metodo che non riempie il buffer ma utilizza solo nuovo dato in ingresso
7. Configuration selection (FSR, f_sampling acc e gyro)
8. ISPU upload an sensors testing

Challenge Introduction

MLC

Build a simple algorithm to recognize 4 classes

Stationariety, Walking, Running and Ciclying

- Analyse a given dataset
- Choose a ML algorithm
- Train and test the model

ISPU

Algorithm fine tuning to embed it in an ISPU

Adjust it to perform real time inference on a sensor

- Tmax = 1 s = 1/ODR
- Pmax = 32768 Bytes
- Dmax = 8192 Bytes

Score evaluation:

$$S = Wa * A + Wt * \left(1 - \frac{T}{Tmax}\right) + Wp * \left(1 - \frac{P}{Pmax}\right) + Wd * \left(1 - \frac{D}{Dmax}\right)$$

Dataset Preprocessing

Original Dataset (sampling frequency of 104 Hz) :

A_X [mg]	A_Y [mg]	A_Z [mg]	G_X [dps]	G_Y [dps]	G_Z [dps]	LABEL
75.281395	961.36255	125.63254	-1.5753003	-0.17680435	0.6350166	1
54.902405	986.3348	147.04503	-1.0415043	-0.0028825735	-0.41466886	1
63.012875	986.5842	127.11164	0.030373422	0.8637093	-0.8405129	1
62.889854	976.0868	114.84553	-1.0292073	-2.4130135	-1.4496281	1
74.71641	995.17365	139.92459	-1.0709769	-1.8179295	-0.4868429	1
78.45848	983.01263	130.82722	0.50987506	1.1800284	-1.1123568	1
68.63823	990.975	115.521835	-1.29094	0.019419545	1.9531794	1
68.50595	983.2555	134.48204	0.32255554	0.9501824	-0.06258155	1
66.38766	982.8169	142.57394	-0.4149284	0.33001274	-0.4774428	1
77.5212	993.1395	135.43755	0.9856307	1.2526073	0.979909	1
85.17584	993.08795	143.45128	-1.0669525	1.5181329	-1.6780736	1
-1197.13	80.7036	33.0143	4.593345	-29.849611	182.0453	2
-1192.7745	98.00611	42.16509	34.546436	-30.30926	183.3217	2
-1144.4713	160.37572	23.784721	64.134705	-29.814598	177.93074	2
-1120.8403	184.13516	2.8536317	82.81598	-29.363989	181.10297	2

.

1. Added new column with Accelerometer and Gyroscope punctual squared norm (vector length) [1 each sample]

$$||n||^2 = x^2 + y^2 + z^2$$

Different combination repeated in the dataset:

#1	#2	#3	#4
1	4	3	1
2	3	1	4
3	2	2	2
4	1	4	3

Dataset Preprocessing

Process data over a time window of 1 second set by the ODR, and offset of 30 samples

2. Average over the time window of each punctual input feature:

$$\text{Mean } \bar{x} = \frac{\sum xi}{N}$$

+

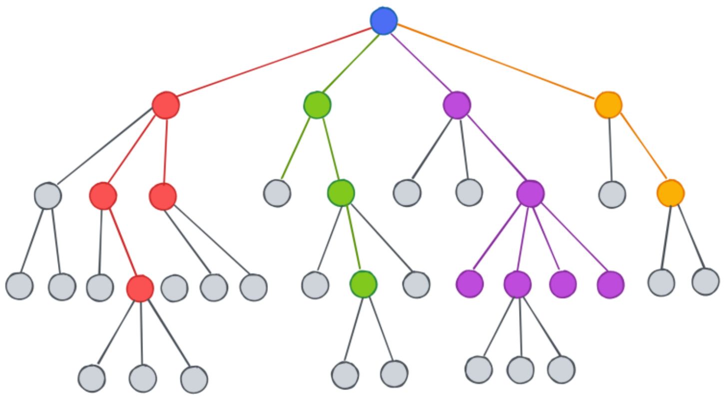
G_Z [dps]	var_A_X [mg]	var_A_Y [mg]	var_A_Z [mg]	var_G_X [dps]	var_G_Y [dps]	var_G_Z [dps]	acc_norm	gyro_norm	LABEL
-0.66233771209…	807.39381058294…	7120.713758159457	6225.350665946…	575.0978396817898	520.48295176126…	52.53842351526…	1011611.224…	1327.9744308…	1
1.989217012519…	15028.432801633…	9048.989752966028	6844.156484943…	767.973240359892	1060.8290958295…	69.89386841026…	1012867.456…	2669.0903904…	1
6.389865639423…	32585.277309740…	13451.62305168954	9677.036876367…	1463.6180121319…	602.805700501188	91.22464106559…	1014820.563…	3905.8234213…	1
12.29390281096…	29694.003361771…	22389.6723171745…	26324.99827889…	1059.687542477981	211.37586197505…	76.78958941524…	1000731.703…	4346.3851847…	1
17.60758955807…	4141.0913440174…	18795.4409512885…	23803.44340899…	1141.8557384991…	130.06745053388…	118.9984209739…	1035199.095…	3887.1826962…	1
16.03180715807…	5212.1904106592…	16148.2160148496…	8049.511253856…	1105.2450686106…	219.35485161496…	256.6763489990…	998335.1935…	2976.7134191…	1

3. Variance over the time window for each input data :

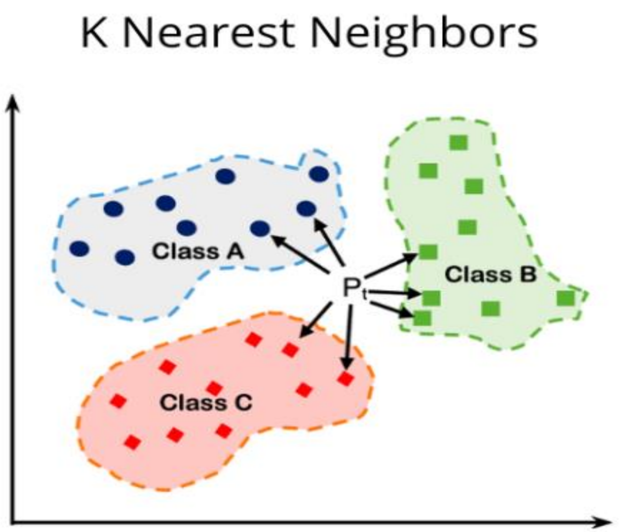
$$\text{Variance} = \frac{\sum (xi - \bar{x})^2}{N}$$

Final input dataset: 14 features:
6 original, 6 variance on the axis, Acc and Gyro norm

Decision Tree vs K-NN



Both NON parametric



Easy to implement and faster inference

Fewer thresholds needed w.r.t k-nn clusters

High overfitting risks

Faster training

Need to upload the whole dataset to perform the inference

Can compute smaller clusters using centroids

Space for libraries needed to perform the euclidean distance, that are also time consuming

Can substitute Manhattan and Hamming distance

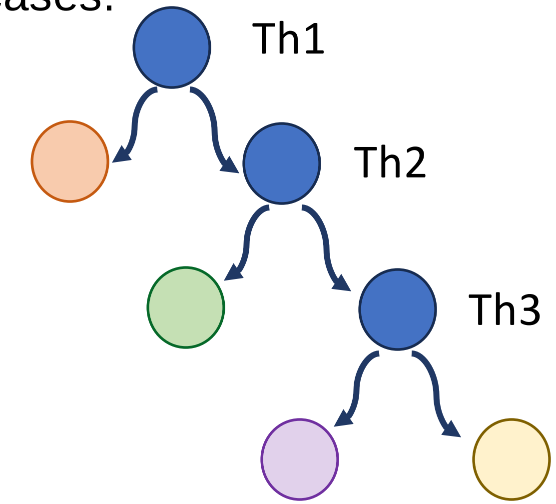
Feature Importance

- We created a model without any restrictions on the features to use and the parameters to create it.
- Feature importances:

[0.03 0.00 0.02 0.00 0.00 0.00 0.26 0.01 0.01 0.00 0.00 0.00 0.32 0.27]

Here we choose to use less features (only 3), neglecting those with low importance compared to the higher ones to avoid OVERFITTING, brought by rare decision that only work for specific cases.

- With this result we decided to select only three features:
  - Variance of the accelerometer on the X axis;
  - Norm squared of the accelerometer;
  - Norm squared of the gyroscope;



Parameter Selection

- After having set the features to use, we decided to do a Grid Search in order to find the most effective parameters.

```python
param_grid = {
    'max_depth': [2, 5, 6, 7, 8, 9],
    'min_samples_leaf': [2, 5, 8, 10, 20 ,30],
    'min_samples_split': [5, 8, 10, 20, 100],
    'ccp_alpha' : [0.001, 0.01, 0.015 ],

}
# Initialize GridSearchCV
grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5, scoring='accuracy')
```

- Then we used a skyline algorithm based on accuracy and number of thresholds to filter the best results:
  - We then decided to opt for the 5th result, since has an optimal ratio between accuracy and threshold counts.

params	accuracy	threshold_count
Depth: 9, Min Leaf: 2, Min Split: 5, Alpha: 0.001	0.9378614859854664	18
Depth: 7, Min Leaf: 2, Min Split: 5, Alpha: 0.001	0.936378466557912	15
Depth: 5, Min Leaf: 2, Min Split: 5, Alpha: 0.001	0.93415393741658	10
Depth: 5, Min Leaf: 2, Min Split: 5, Alpha: 0.01	0.9262939344505412	3
Depth: 2, Min Leaf: 2, Min Split: 5, Alpha: 0.001	0.7179297048791339	2

Model Selection

- Based on the features and the parameters selected, we created a model and made a function that translates the created tree(.py) into a pseudo-C code, to make it easy to create the code for the sensor.

```python
def tree_to_code(tree, feature_names):
    tree_ = tree.tree_
    feature_name = [
        feature_names[i] if i != _tree.TREE_UNDEFINED else "undefined
        for i in tree_.feature
    ]

    def recurse(node, depth):
        indent = "  " * depth
        if tree_.feature[node] != _tree.TREE_UNDEFINED:
            name = feature_name[node]
            threshold = round(tree_.threshold[node], 4)
            print(f"{indent}if ({name} <= {threshold}f) {{")
            recurse(tree_.children_left[node], depth + 1)
            print(f"{indent}}} else {{ // if {name} > {threshold}f")
            recurse(tree_.children_right[node], depth + 1)
            print(f"{indent}}}")
        else:
            values = tree_.value[node][0]

            for i, val in enumerate(values):
                print(f"{indent}decision[{i}] = {int(val)};")

    recurse(0, 0)
```

```c
if (acc_norm <= 3085268.25f) {
  if (gyro_norm <= 6713.5347f) {
    if (var_A_X [mg] <= 6166.4956f) {
      decision[0] = 5860;
      decision[1] = 63;
      decision[2] = 0;
      decision[3] = 55;
    } else { // if var_A_X [mg] > 6166.4956f
      decision[0] = 507;
      decision[1] = 332;
      decision[2] = 0;
      decision[3] = 6088;
    }
  } else { // if gyro_norm > 6713.5347f
    decision[0] = 291;
    decision[1] = 6859;
    decision[2] = 47;
    decision[3] = 424;
  }
} else { // if acc_norm > 3085268.25f
  decision[0] = 0;
  decision[1] = 0;
  decision[2] = 6440;
  decision[3] = 5;
}
Feature importances:  [0.29769464 0.38172859 0.32057677]
Accuracy: 0.9368233723861783
Threshold Count: 3
```

Porting on the ISPU

Setup & Reading Data

- Our solution is implemented in the algo_00, where in the _init phase we call the function reset_status() where we initialize all the variables used in the code.

```
// reset window counter and accumulators
void reset_status(void)
{
    win_count = 0;
    gyro_norm_sqred = 0.0f;
    acc_norm_sqred = 0.0f;
    a_x_k = 0.0f;
    a_x_ex = 0.0f;
    a_x_ex2 = 0.0f;
}
```

- In the algo_00 itself we start incrementing the window count and we read all the data from the sensor

```
void __attribute__((signal)) algo_00(void)
{
    win_count++;

    // read data
    float a_x = (float)cast_sint16_t(ISPU_ARAW_X) * ACC_SENS;
    float a_y = (float)cast_sint16_t(ISPU_ARAW_Y) * ACC_SENS;
    float a_z = (float)cast_sint16_t(ISPU_ARAW_Z) * ACC_SENS;
    float g_x = (float)cast_sint16_t(ISPU_GRAW_X) * GYR_SENS;
    float g_y = (float)cast_sint16_t(ISPU_GRAW_Y) * GYR_SENS;
    float g_z = (float)cast_sint16_t(ISPU_GRAW_Z) * GYR_SENS;
```

Porting on the ISPU

Processing Data

- We implemented the feature extraction pipeline, computing the mean of the squared norms and the variance of the x axis of the accelerometer with inline algorithms in order to optimize the size of our solution, without the need to store large set of data.

```c
//calculate norm_squared
float a_norm_sqred = a_x * a_x + a_y * a_y + a_z * a_z;
float g_norm_sqred = g_x * g_x + g_y * g_y + g_z * g_z;
//calculate norm mean
acc_norm_sqred  += (a_norm_sqred - acc_norm_sqred) / win_count;
gyro_norm_sqred += (g_norm_sqred - gyro_norm_sqred) / win_count;

//setup variance acc_x
if (win_count == 1)
{
    a_x_k = a_x;
}
a_x_ex += a_x - a_x_k;
a_x_ex2 += (a_x - a_x_k) * (a_x - a_x_k);

int16_t prediction = 0;

if (win_count == WIN_LEN_IN_SAMPLES)
{
    float var_a_x = (a_x_ex2 - (a_x_ex * a_x_ex) / win_count) / (win_count - 1);
```

Memory result:

```
reisc-size bin/ispu
   text    data     bss     dec     hex filename
   3132      96      28    3256     cb8 bin/ispu
```

D = 124 Bytes
P = 3132 Bytes

Porting on the ISPU

Output prediction

- When we reach the number in samples to fill our window, then we proceed making the prediction.
- We then send our prediction into register ISPU_DOUT_00 .
- We modified the .json file in order to see the output correctly.

```
{
    "output":
    [
      {
        "name": "Predicted Activity",
        "type": "int16_t"
      }
    ]
}
```


```
if (win_count == WIN_LEN_IN_SAMPLES)
{
    float var_a_x = (a_x_ex2 - (a_x_ex * a_x_ex) / wir

    if (acc_norm_sqred <= 3085268.25f)
    {
        if (gyro_norm_sqred <= 6889.0322f)
        {
            if (var_a_x <= 6461.0833f)
            {
                prediction = 1;
            }
            else
            { // if var_A_X [mg] > 6461.0833f
                prediction = 4;
            }
        }
        else
        { // if gyro_norm > 6889.0322f
            prediction = 2;
        }
    }
    else
    { // if acc_norm > 3085268.25f
        prediction = 3;
    }

    cast_sint16_t(ISPU_DOUT_00) = prediction;

    reset_status();
}

// interrupt generation (set bit 0 for algo 0, bit 1 f
int_status = int_status | 0x1u;
```

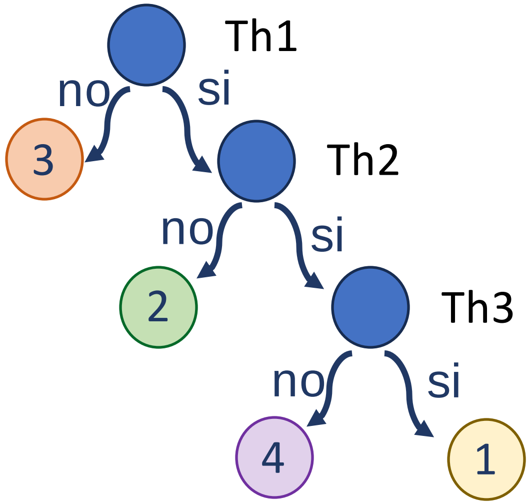
Encountered problems

- Overfitting -> feature extraction + simpler models

```
Feature importances:  [0.26415994 0.31125496 0.30835614 0.0284965  0.04847023 0.03926224]
Accuracy: 0.9424141749723145
Threshold Count: 1067
```

- Configuration settings. We changed Gyro FSR from 2000 to 1000 dps to improve the resolution and we solved ambiguity to distinguish between class 2 and 4.

Results




```
Accuracy: 0.8773063624907099%
```

A = 0.8773
T = 590 us
D = 124 Bytes
P = 3132 Bytes

$$S = Wa * A + Wt * \left(1 - \frac{T}{Tmax}\right) + Wp * \left(1 - \frac{P}{Pmax}\right) + Wd * \left(1 - \frac{D}{Dmax}\right) = 93.13$$

Thanks for your attention

life.augmented