Identification of the Best Accelerometer Features and Time-scales to Detect Disturbances in Calves

Oshana Dissanayake, Sarah McPherson, Emer Kennedy, Katie Sugrue, Muireann Conneely, Laurence Shalloo, Pádraig Cunningham and Lucile Riaboff

Oshana Dissanayake

Ph.D. Student – University College Dublin, Ireland - Insight

7th Workshop on Advanced Analytics and Learning on Temporal Data (AALTD@ECML)

23rd September 2022

A World Leading SFI Research Centre

Science Foundation Ireland For what's next

HOST INSTITUTION

PARTNER INSTITUTIONS

FUNDED BY:

An Roinn Talmhaíochta, Bia agus Mara
Department of Agriculture, Food and the Marine

Digitalising Dairy

Background

- Development of new decision tools in the context of Precision Livestock Farming is booming in the research field at present but lack of tools to detect stressful events or diseases, especially in calves.

- Early detection of health and welfare disturbances would help to increase cattle welfare and decrease the cost of treatment of cattle.

- Automatic early detection of stressful events in calves would be a major contribution to the field.

- For that It is required to identify the best accelerometer features that can be used to detect disturbances in calves, which is our objective in this study

Methodology – Data Preparation & Feature derivation

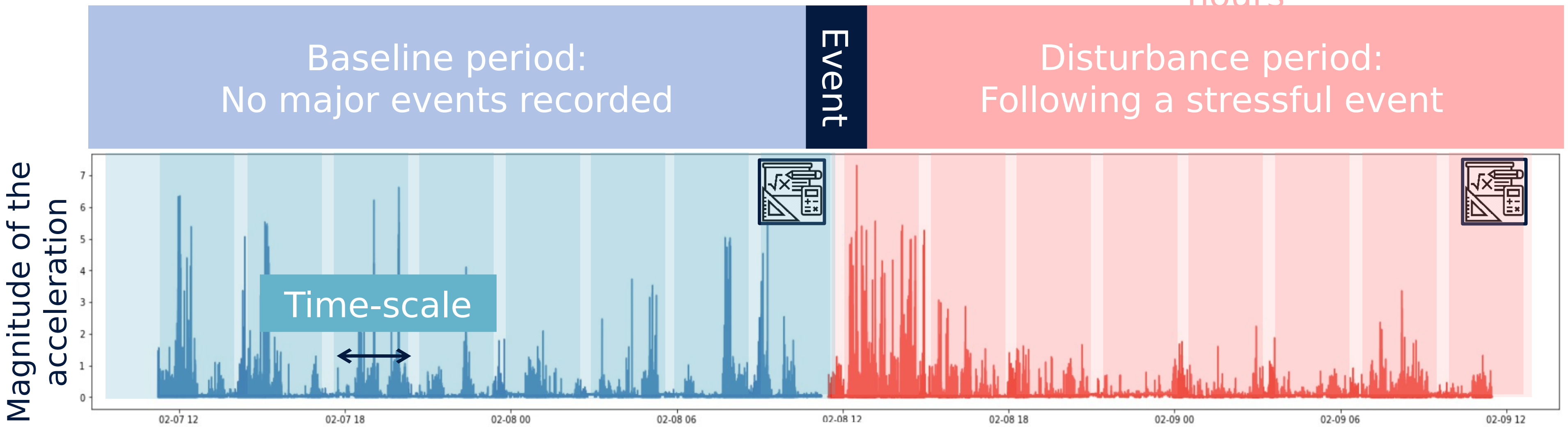
Teagasc
Agriculture and Food Development Authority

Accelerometer sensor (25 Hz)

47 calves

Axivity accelerometer sensor

- 24 hours
Dehorning
+ 24 hours



1 second ⟶ 24 hours

Time

Energy expended

Mean
Median
⟶ Overall level

Motion Variation
St. Deviation
⟶ Dynamic behaviors

⟶ Maximum
Activity peaks

Spectral entropy
Fractal features
⟶ Structure complexity

Activity structure

VistaMilk

Methodology – Feature Reduction Pipeline & Performance Evaluation

Fea 103 features Ti scales

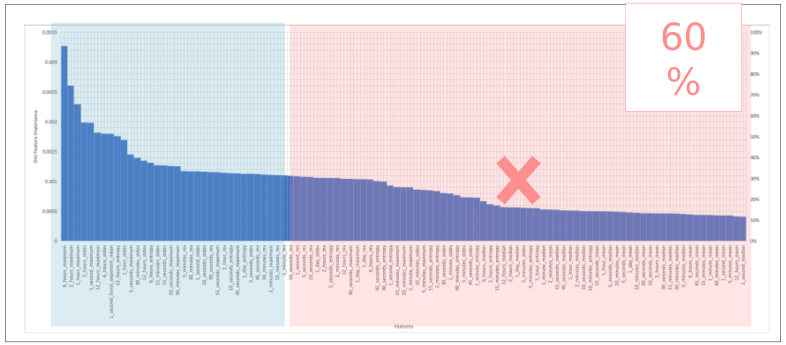
47 calves

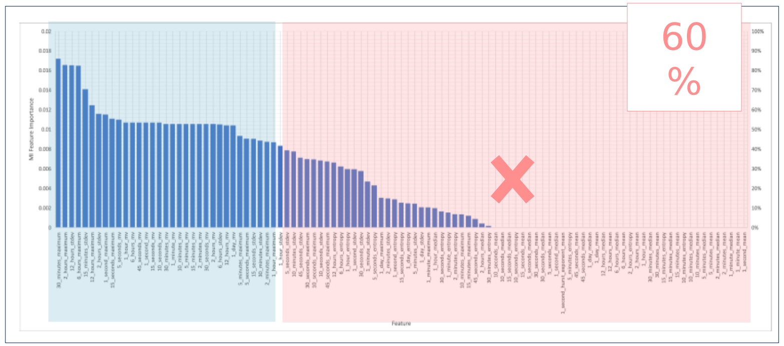
Data split	Initial Filter		Second Filter		ML model
Train 60%	Mutual Information (MI)	40 MI features	Random Forest Feature Importance (RFFI)	14 RFFI + MI features	Random Forest
Validation 20%					Gaussian Naïve Bayes
Test 20%	Gini Index	40 Gini features		8 RFFI + Gini features	K-Nearest Neighbor

Initial Filter: 10 iterations

Second Filter: 10 iterations

ML model: 10 iterations





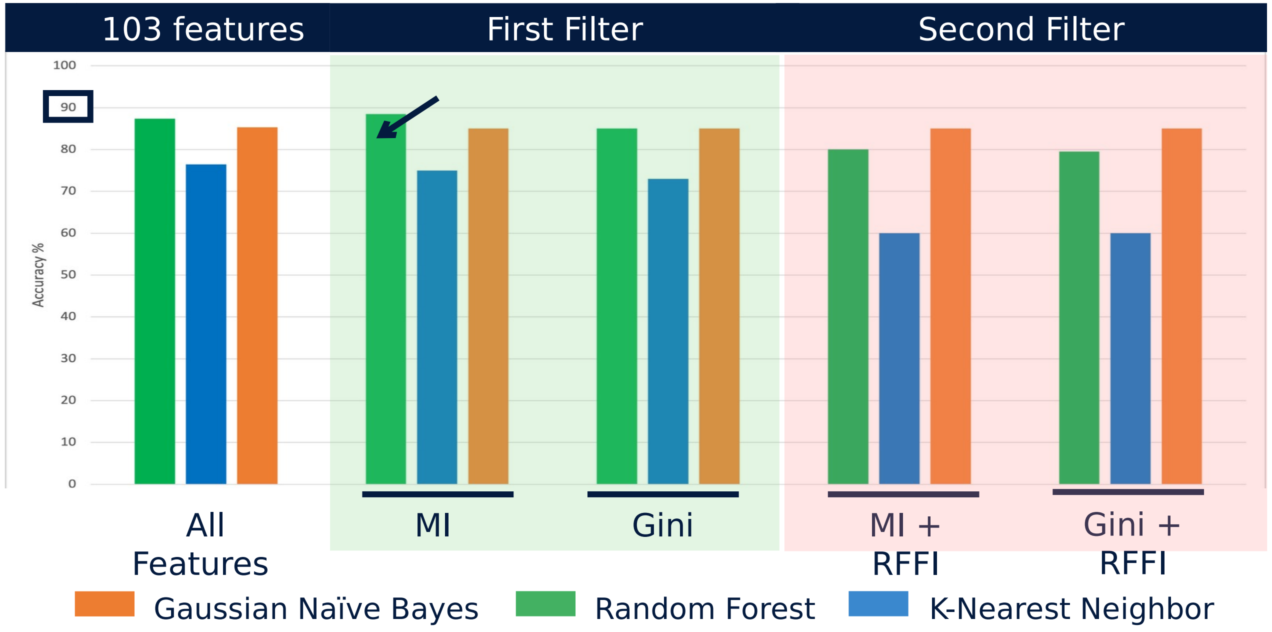
VistaMilk

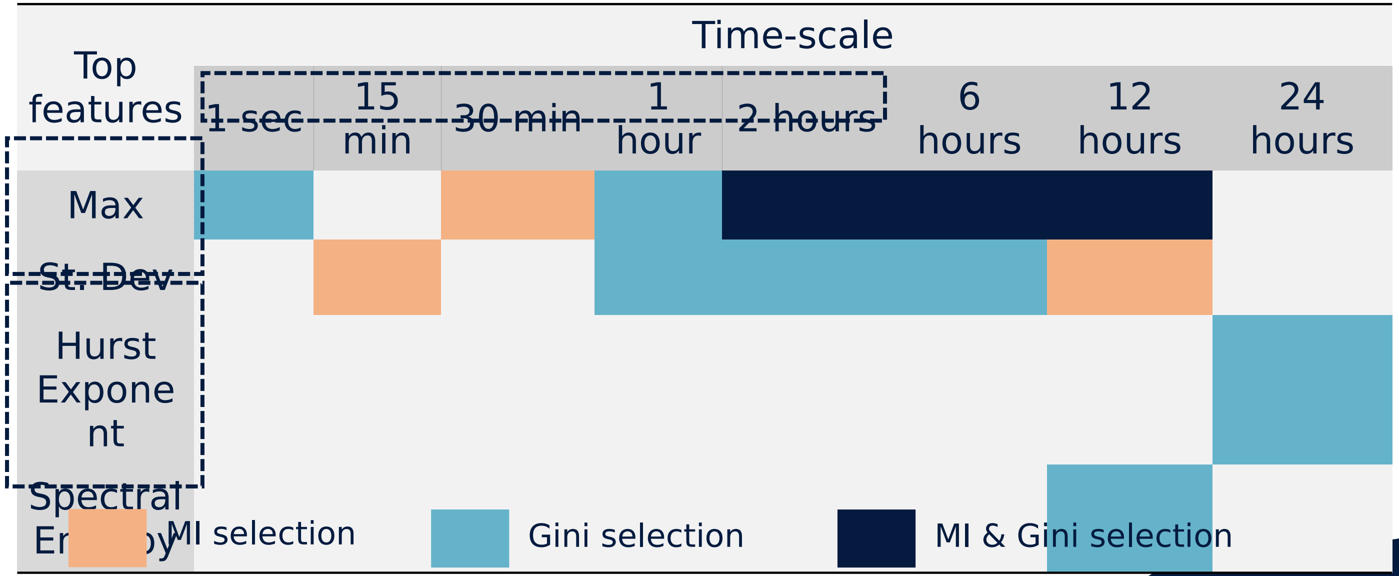
Results

1 The highest accuracy of 90% with Random Forest with features selected using MI.

2 Random Forest Feature Importance (RFFI) results in a reduction in accuracy – probably due to overfitting. So better to rely on MI & Gini selection to find the best features & time-scales

3 Best features < Energy expenditure & activity structure
Best time-scales ϵ [1 sec ; 24 h]





Thank You

Further Information

Oshana Dissanayake
Insight Centre for Data Analytics – University College Dublin, Ireland
Email: oshana.dissanayake@ucdconnect.ie

A World Leading SFI Research Centre
Science Foundation Ireland For what's next
HOST INSTITUTION
Teagasc Agriculture and Food Development Authority
PARTNER INSTITUTIONS
FUNDED BY:
An Roinn Talmhaíochta, Bia agus Mara
Department of Agriculture, Food and the Marine
Science Foundation Ireland For what's next

Spectral Entropy

Libraries:

SciPy

$$SE(F) = \frac{1}{logN_u \sum_u (P_u(F) log_e P_u(F))}$$

```
# to calculate the entropy

# window_Amag = The signal-window for which you want to compute the spectral entropy
# WS =  The window size in seconds
# sampling_rate = The sampling rate expressed in Hz (12.5 Hz)

# the reason for using len(window_Amag) for the hamming function: The shape between the
# window_Amag size and the WS*sampling_rate size requires to be the same for the fft function to calcuate the
# transform. But in some windows, the missing data makes it unequal giving errors. The inequality
# occurs with minor margins. Thus for the fft function len(window_Amag) is used and for the
# absAmagFFT WS*sampling_rate is used because when selecting
def calculate_entropy(data,sampling_rate,WS):

    window_Amag = np.array(data)

    # fft using Hamming Window
    ## AmagFFT = fft(window_Amag*np.hamming(WS*sampling_rate)) ##
    AmagFFT = fft(window_Amag*np.hamming(len(window_Amag)))

    # Single-sided spectrum = module fft
    absAmagFFT = abs(AmagFFT[1:math.floor(WS*sampling_rate/2+1)])**2

    WeightAbsAmagFFT = absAmagFFT/sum(absAmagFFT)

    SpectralEntropy = -sum(WeightAbsAmagFFT * np.log(WeightAbsAmagFFT))

    return SpectralEntropy
```

Motion Variation

MV = mean(abs(1st order discrete difference of the signal window))

```
def calculate_motion_variation(window_Amag):
    mv = statistics.mean(abs(np.diff(window_Amag)))
    return mv
```