Detect chewing episodes with an Inertial Measurement Unit (IMU) sensor around the ear

Vivian Nguyen

Supervisors: Przemysław Pawełczak, Vivian Dsouza

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Vivian Nguyen
Final project course: CSE3000 Research Project
Thesis committee: Przemysław Pawełczak, Vivian Dsouza, Hayley Hung

Abstract

Tracking food intake provides a valuable source of infor- mation to gain insights in dietary habits for the health indus- try. Currently, the main method to track food intake to is do it manually. To take a step towards tracking food intake manually, this these aims to answer the following research question: "How to detect chewing episodes with an IMU sensor around the ear?". The IMU collect x, y, z axis data for the accerelometer an gyrscope. The data of the axis was downsampled to 20Hz and passed through a low-pass filter. Chewing detecting was done by determining chewing episode in time windows of 30 seconds. Feature were extracted from those time windows. Then random forest, decision tree, k-neareast neigbours, support vector machine and logistic regression machine learning models were used to classify the data, for which f1-scored higher than 0.8 have been achieved. Therfore it can be concluded is it posisble to detect chewing episodes with a single IMU around the ear. Feature selection and performance analysis has been done, and it seem to be that features that use auto correlation and Fast Fourier Transform (FFT), play a significant role increasing the classification performance. They are computationally expensive and are not ideal for embedded system. However there is room for finding features that are less computationally expensive.

1 Introduction

Tracking food intake provides a valuable source of information to gain insights in dietary habits for the health industry. The health industry can utilise this information, to better understand a persons well-being and find a more suitable diagnosis or treatment. Another example is to track whether daily nutrients or medicines have been taken. Tracking food intake results in measurements of the amount and type, the time and duration of consumption and the nutritious value of the food taken.

Currently, the main method to track food intake to is do it manually. A potential new method could be to make use of *earables*, "devices that attach in, on, or in the immediate vicinity of the ear to offer functionalities beyond basic audio in- and output." [8]. Earables are ideal for the following reasons. Firstly, because an earable is placed around the ear, sensors can better pick up movements that happen during eating. Secondly, the adoption of earables can be easier and faster since headphones, earphones and earbuds are already part of everyone's daily life. And lastly, automatically tracking food intake seems more desired, as opposed to manual tracking, as it provides convenience and possibly more precision.

Detecting food intake consists of multiple steps. The first step is to determine whether a user is taking in food, see figure 1. Chewing detection is part of this first step, and the main focus of this thesis. Previous thesiss have done research in detecting chewing with sensors such as IMU's, microphones, piezzo-electric sensors and more. Experiments have been done with various combinations of sensors at different locations of the body. However no research has been done with a *single IMU sensor that was placed around the ear* to detect chewing episodes.

The main research question for this thesis is:

"How to detect chewing episodes with an IMU sensor around the ear?".

With an earable being an embedded system, a section will be dedicated detecting chewing episodes with a simple low-power algorithm for an embedded platform with a single IMU sensor around the ear.

This thesis will explain the methodology and the experiments that were conducted to answer the research question. The results that were obtained, as well as the conclusions that were derived from the results will be presented. At the end of the thesis future recommendations and improvements are provided

2 Related work

Various researches have been conducted in the past and thesiss are published on experiments related to chewing detecting.

Various methods

Chewing detection has been done by counting the chews, as researched by M Farooq and E. Sazonov [4].

Another way of detecting chewing was by determining whether chewing was happening in a certain time frame, investigated by Abdelkareem Bedri et al. [1] and Roya Lotfi et al. [5].

Temiloluwa Prioleau et al. [7] and N. A. Selamat and S.H.Md. Aki [9] have analyzed articles that perform chewing detection and have made side by side comparisons. A lot of different sensors and combinations of sensors have been used, such as a piezzo-electric senso [4] [3], microphone [5], IMU [1] [5]. Temiloluwa et al. [7] have depicted a clear diagram of the steps of detecting food intake in their thesis. See image 1 for this diagram. Notice that this thesis focuses the first part, 'chewing'.

In R.R.Choouhury's paper, opportunities of earables are explored on a surface level. Also, the question has been raised whether "today's hardware architecture mostly suffice, or is a new clean-slate thinking necessary." [2]. Is it important to think about this at a future stage, when chewing detecting integrated into an earable. User experience is also a crucial aspect to consider, since that heavily determines the use of a device or product.

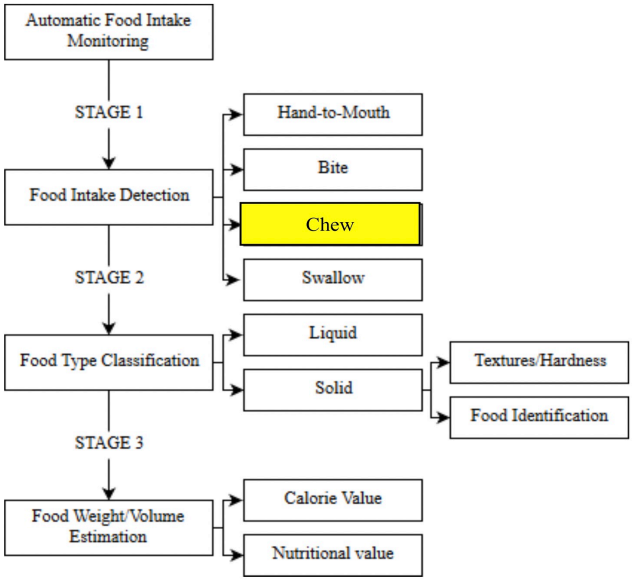
Similar setup

Both Abdelareem Bedri et al. [1] and Roya Lofti et al.[5] use IMU sensors and have a similar approach to this thesis. Both thesis detect chewing episodes using time windows to classify chewing. Interestingly Abdelkareem Bedri et al.[1] uses 30s 1-sec-sliding time windows, and Roya Loft et al.[5] use 3s non-overlapping windows. Both used video recordings as the ground truth to label the data.

Abdelkareem Bedri et al[5] uss an IMU and a microphone on a device ”in” the ear”, like a earphones. They have used logistic regression, decision tree and random forest as classification methods. They concluded that using the IMU data better detected chewing for soft and hard foods, as compared to microphone data, and that the combination of both sensors data worked best.

Roya Lofti et al.[1] use an IMU sensor behind the ear, on the temporalis muscle to be more precise. With 13 features for each axis of collected data, they used sequential forward floating selection to make a feature selection, and applied the random forest classification model. An extra IMU on the back was used to cancel out ”large body motions”. With this setup of the two IMUs for their ”outside-the-lab-study”, it raises the question how only one IMU behind the ear will perform, which this thesis aims to answer.

Figure 1: ”Overview of automatic food intake monitoring system” [9], (fig. 1)



3 Methodology

To answer the research question the methodology to conduct the research can be split into two parts. The first part discusses how the data was gathered with the IMU sensor and explains the classification method to classify the data into chewing or not chewing. The second part will be dedicated to detecting chewing episodes with a simpler low-power algorithm for an embedded platform with a single IMU sensor around the ear.

3.1 Part 1: Experiment

Hardware components and software

The IMU is the Qwiic 9DoF IMU (ICM-20948). This is an IMU that measures x, y, z axis rotations with its gyroscope and x, y, z axis accelerations with its accelerometer. The micro-controller to which the IMU is attached is the NUCLEO-L433RC-P.

The programming language that was used is Python. Python is commonly used for data processing and machine learning and has many useful libraries that can be used. The C language would be ideal in case of using an IMU with the micro-controller on an embedded system.

With the IMU attached to the micro-controller, which itself is attached to a computer with a data and power transmittable cable, the IMU data will be collected on the computer. The computer will run the needed algorithms written in Python to analyse the data.

Data collection

The data should be classified into chewing and not chewing, thus data needs to be from activities from both classes. The chosen activities are eating, walking, reading out loud, and stationary activities such as watching a video or studying. The collected data will be labeled into two classes. Only the eating activities will be labeled as ’chewing’. All the other activities will be labeled as ’not chewing’.

Feature selection

Inspired by the Abdelkareem Bedri et. al [1] and Dan Morris et al. [6], the following features were chosen to extract from the data:

- Standard deviation
- Mean
- Variances of the duration between each zero-crossing
- RSM of the power signal
- The number of zero-crossings
- The variance of the power signal
- The number of peaks after autocorrelating the data
- The number of prominent peaks after autocorrelating the data
- The number of ”weak” peaks after autocorrelating the data
- The value of the max peak after autocorrelating the data
- The first peak after the first zero-crossing after autocorrelating the data
- Dominant frequency, after applying FFT
- Value of dominant frequency, after applying FFT

With 6 axis x 13 features = 69 total features for each object in the dataset. Sequential Forward Selection (SFS) will be done to do select a set of most influential features do decrease the feature set size. The SFS will provide an approximation of a set of features that give the highest performance.

Data processing

To classify the gathered data over the measured time period, a 30 second window will be used for which the selected features will be computed. The window will be labeled as either chewing or not chewing, assigned to the class that happened the majority during the window timeframe. Inspiration was

taken from Abdelkareem et al. [1] where time windows of 30 seconds with a slide of 1 sec were used. In a performance graph they showed that 35 seconds window performed most optimal in their experiment. However to the fact that data collection was easier done in 30 second, 30 seconds time windows were chosen. And to minimize the sample windows to process, the choice was made to do use tumbling instead of sliding windows.

3.2 Part 2: Analysis

The second part will be dedicated to analysing and considering the required factors to think about when integrating chewing detection in an embedded system

With the code written in Python and it being run on a computer, it is difficult to say how the algorithm would behave on an embedded system. This is due to the fact that another lower level programming language such as C, which is more suitable for embedded systems, may have a faster runtime. The embedded system may also have a different memory space, as well as different battery life and number of CPUs. If implemented in an earable, which also provide other functionalities, a balance should be found in how each of those use the runtime, memory space and power.

No measurements will be done on how much memory or clock cycles the code uses. Nevertheless, to accommodate further research in how chewing detection can best be implemented in an embedded system, alteration of the classification method have been done and their performance is computed. When working towards a working embedded system, being attentive to the used memory space, runtime, and battery life, the classification performance can be taken into account to determine which trade-offs to make. As well as the time of computation, energy consumption and memory it takes to store and compute the algorithm. Especially if the chewing of food intake detecting is to be integrated into an earable, which also has different functionalities.

4 Experiment

This section discusses how the experiment was conducted.

4.1 Sensor setup

The chosen location of this sensor was above and behind the left ear, on the temporalis muscle, as in [1]. Since only only one sensor was in my possession, the left ear had been chosen. Because the wires were not so long and the ports on my laptop are on the left side. The IMU sensor is not integrated into an earable/wearable device. Thus creative ways were needed to attach the sensor to the allocated location. One solution was sticking it with tape to the skin. A downside of this is that it was not comfortable while doing the chosen activities. When removing the sensor, the skin got sensitive and got hair stuck to it. Another solution was sewing the IMU sensor to the hairband. This was a lot more comfortable and useful, and thus was the actual used setup. [see image 2]

Since the sensor had to be attached to the laptop, it was difficult to move unrestrictedly. Because of this, it was hard to do the chosen activities continuously. To simulate as if the activities were done continuously, the activities were done one by one, and later merged into one data set.

Figure 2: Sensor setup, sewn in headband

4.2 Data preparation

After an activity was measured with the worn sensor, the data of the 6 axis were written to a textfile. See the table 1 for the activities that were measured during the experiment. If a similar activity was done or item was eaten, the sensor was reattached to such the the session could be marked as a seperate activity. This allows the data set and test set to consist of different data. Table 2 shows the performance of all chosen machine learning algorithms for different train-test set splits. The performance was achieved by tuning the hyperparameters with gridsearch on the f1-score.

The data was processed in the following order: downsampling to 20Hz, applying the low filter for each axis, moving axis data to oscillate around the 0-axis, section the data set into time windows of 30 seconds and compute features. For which the low-pass filter parameters were fs=20 and Wh=4, and the autocorrellation had a lag of 250. See figure 3 for the pipeline and see figures 4, 5, 6, 7 for an example of some of the pipeline steps. It shows the downsampling, low-pass filtering, autocorrelating, and finding the number of peaks after autocorrelation.

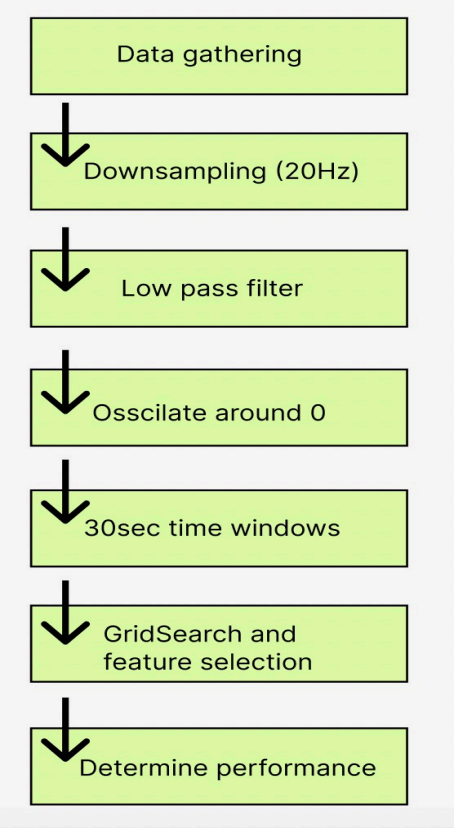
4.3 Data classification

The chosen machine learning models that were applied are random forest (RF), decision trees (DT), logistic regression (LR), support vector machines (SVM) and k-nearest-neighbors (KNN) to do the binary classification. Tumbling

Table 1: Activities per person

Activities per person				
	Eating	Talking	Walking	Studying/Video/Phone scrolling
P1	24.5 min	11 min	11 min	24 min
P2	5,5 min	8 min	9 min	12 min
P3	2 min	4 min	7 min	5 in
P4	6 min	3,5 in	4 min	5 min
total	34 min	26.5 min	31 min	46 min

Figure 3: Pipeline algorithm



time windows of 30 have been used. The scikit python library was uses. Gridsearch, an algorithm in scikit that finds the optimal set of hyperparameters was used to tune the machine learning models. Table 1 shows that chewing/eating is approximately 25% of all the data. Due the the imbalanced data set, the Gridsearch performance was based on the f1-score. Precision and recall is needed to compute the f1-score, as opposed to the accuracy, where if all the data gets classified to the biggest class, an accuracy of 75% would be achieved. See appendix A for the used Gridsearch parameters per algorithm.

4.4 Analysis

The following change has to be made to the machine learning pipeline: Decrease the amount of features used. Use sequential forward feature selection (SFS) to determine which

Figure 4: Example of pipeline steps for detecting number of auto-correlation peaks, for a 30sec time window on an arbitrary axis when eating a cookie.

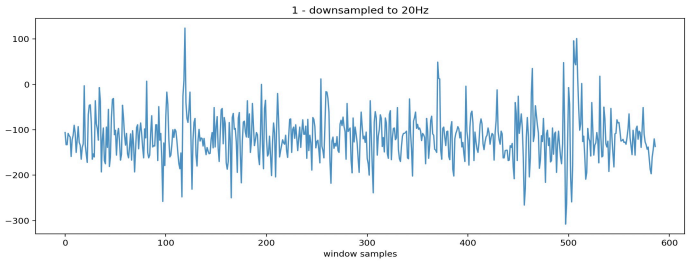
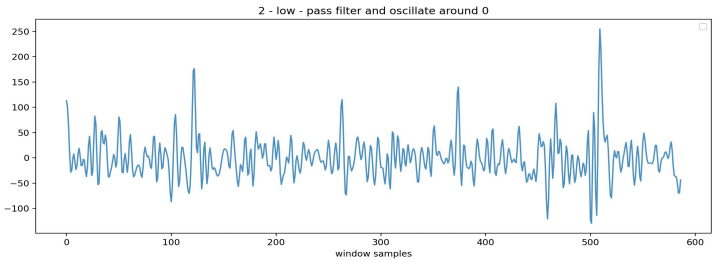


Figure 5: Example of pipeline step for detecting number of autocorrelation peaks



feature sets to use and determine which features are most important to the classifcation. Sets of size 1, 2, 3, 5, 10, 20, and all features have been computed using SFS Feature sets were also computed consisting of all except the auto-correlation features, all except the FFT features and all except both FFT and autocorrelation features. Besides the computation time and memory it cost to compute the features, putting the new data into the trained classification model in the embedded system has to be considered too.

5 Results

See table 2 for the performance of the chosen algorithms. The performance was measured by the accuracy and the f1-score. The split of the train and test set are also shown. 4 splits were done for the train and test set and tested on windows with all features. Various splits resulted to different performance scores. The first two splits are most sensible data splits, where training has been done on various participants, and testing on an unknown participant, with a roughly 75%-25% split. Looking at these two splits, Random forest

Figure 6: Example of pipeline step for detecting number of autocorrelation peaks.

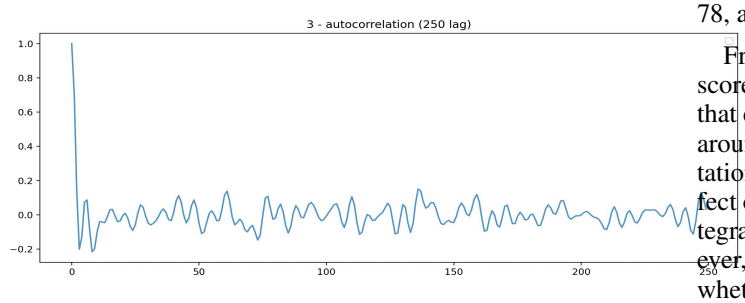
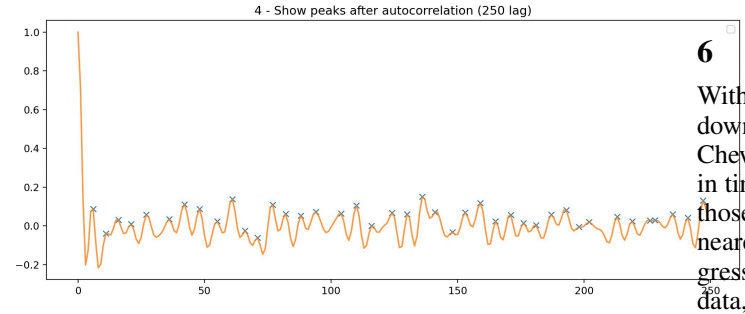


Figure 7: Example of pipeline step for detecting number of autocorrelation peaks



performs has the best f1-score. Testing on p2 gave a lower peformance than on testing on p3 and p4 together. Perhaps the data of p2 looked a lot more different than those of the other participants, due to different body motions.

With the second split, more analysis have been conducted. One extra analaysis has been done where sample windows had all features, all except features that use autocorrelation, all except features that use FFT and all except features that use autocorrelation and FFT. This was done to see the impact of autocorrelation and FFT features, since both are quite computationally heavy. Table 3 shows the performance. Interestingly, random forest with an f1-score of 0.81 and k-nearest neighbors with a f1-score of 0.62, perform better without autocorrelation or FFT features. Overall, the autocorrelation features seem to achieve a higher performance than the FFT. However it is important to remember that there are more features that use auto correlation, which probably impacted this. But if this not take up more space, or time or energy from an embedded system, the many more features autocorrelation has, should be no issue.

With this second train-test split, sequential forward feature selection has been done too. In appendix C, D, E, F, G the selected feature sets can be found. An interesting observation is that the acceleratormeter axis' were the most frequently selected axis to extract a feature from. KNN and DT seem to use the gyroscope data relatively more compared to the other models.

The f1-score for all of the models significantly increases when the feature set goes from 5 to 10. It is also that when going from 5 to 10 features, more FFT or autocorrelation features are chosen. And they mostly only go up when going from 10 to 20 features, where even more FFT or autocorrelation features are selected. With too many features, namely all 78, and somethings already 30, the performance decreases.

From all the performance analysis, with the highest f1-scores being above 0.8, it seems it is possible to conclude that chewing detecting can indeed be done with a single IMU around the ear. Autocorrelation and FFT features are computationally heavy, however they seem to have a significant effect of the classification performance. This is not ideal for integrating chewing detecting into an embedded system. However, there seems to be room for further research and discover whether other features or tweaking parameters can minimize or remove the use of autocorrelation and FFT features, using also more user data.

Also see table

6 Conclusion

With the IMU data has been collected. The axis data has been downsampled to 20Hz and passed through a low-pass filter. Chewing detecting was done by determining chewing episode in time windows of 30 seconds. Feature were extracted from those time windows. Then random forest, decision tree, k-neareast neigbours, support vector machine and logistic regression machine learning models were used to classify the data, for which f1-scored higher than 0.8 have been achieved.

Feature selection with sequential forward feature selection and performance analysis has been done, and it seem to be that features that use auto correlation and Fast Fourier Transform (FFT), play a significant role increasing the classification per- formance. They are computationally expensive and are not ideal for embedded system. However there is room for finding features that are less computa- tionally expensive. Also, with more data from more participants the performance might increase too. However, overall it can be concluded that detecting chewing episode is possible with a single IMU around the ear. And further research may lead to more promising result towards truly detecting food intake in an earable.

7 Improvements and recommendations

Working on this research paper and conducting the experiment have been done in a span of 9 weeks. With this limited time period, points of improvements - that were identified before, during and after the experiment - are to be mentioned, as well as recommendations for future work. The following can be found below:

- With having access to one IMU sensor for one ear, it could be interesting to detect chewing with IMU sensors on both ears, and how this has an effect on the classification, as well as integrating in to an embedded system. Could the sensors alternately do the detecting to save battery or memory, or would one sensor be better, considering the overall usage?
- Considering the time span of the research, data was collected on 4 participants. Collecting and using data from more people can provide a more generalized result.

- Research could be done to find the best place to attach the sensor, as well as a contructed way to attach the sensor at that specific location, such that how the user wears or places the sensor, does not or minimally influence the sensor location.
- This research determines when a user is chewing or not, with time windows. However chewing does not directly correlate with how much someone eats, because people may take different bite-sizes, or take a different amount of chews before they swallow. Determining how much someone chews or eats can be an interesting research topic. This as well as determining what someone eats. (Refer back to figure 1 for food intake steps)
- This research has done performance analysis. The next step would be to actually implement it on the embedded system and see the effects on speed, power and memory. This also rises the question of how much of the earable should be embedded, and how much can actually be done on/moved to an external device, especially assuming that an interface will be needed.
- The data was collected by doing particular activities and adding the data sets together. A more accurate representation of real life daily activities would have been to do the activities after one another, doing the activities simultaneously. A video recording could be used as the ground truth to label the data.
- Different or additional features could be have been chosen, aswell as different (hyper)parameters.
- As this thesis tried to find a low power and low memory algorithm for embedded systems, can this chewing detection be used as a prompt to start further food intake analysis, for those that might be more demanding on the embedded system? Is it possible to find a more lower powered and lower memory algorithm, maybe one that does not use machine learning?

8 Responsible Research

Responsible research is a broad term that discusses among others, the validity of the research and ethical concerns. The following subsections will touch upon these topics for this particular research project.

8.1 Reproducibility

The codebase of the project can be found in a (private) GitHub repository. With both the GitHub repository and this thesis, it is to be believed that enough information is provided such that the experiment can be reproduced.

8.2 Ethical concerns

Especially with the IMU sensor being integrated in a wearable device, ethical concerns need to be considered. With the IMU sensor collecting data from its 6 axis, along with the time of measurements, the collected data during this experiment does not seem to be of a sensitive nature.

At a later stage, if food intake detection becomes highly accurate, other parties might take advantage of this knowledge. For example insurance companies asking for a higher insurance rate if they know someone has unhealthy dietary habits. Or advertisement companies showing more tailored advertisements. It is thus important this private data is protected. If a person feels overall uncomfortable with this collection of data, the possibility should be available to activate or deactivate food intake detection.

8.3 Data

All 4 participants who's data has been collected have consented to participating in the experiment, as well as their data being used for the thesis and it being published. It has to be mentioned that 3 of those participants are involved in this or a similar research project.

8.4 Validation

Regarding the testing and validation of the desired feature extraction, example plots were used to manually verified if it produced the desired output. Some measured activities showed a not working accelerometer z-axis. These data samples were ommitted

References

[1] Abdelkareem Bedri, Richard Li, Malcolm Haynes, Rai Prateek Kosaraju, Ishaan Grover, Temiloluwa Prioleau, Min Yah Beh, Mayank Goel, Thad Starner, and Gregory Abowd. Earbit: Using wearable sensors to detect eating episodes in unconstrained environments. 2017.

[2] Romit Roy Choudhury. Earable computing: A new area to think about. 2021.

[3] Muhammad farooq and Edward Sazonov. Detection of chewing from piezoelectric film sensor signals using ensemble classifiers. 2016.

[4] Muhammad farooq and Edward Sazonov. Linear regression models for chew count estimation from piezoelectric sensor signals. 2016.

[5] Roya Lotfi, Heorge Tzanetakis, Rasit Eskicioglu, and Pourang Irani. A comparison between audio and imu data to detect chewing events based on an earable device. 2020.

[6] Dan Morris., T. Scott Saponas, Andrew Guillory, and Ilya Kelner. Recofit: Using a wearable sensor to find, recognize, and count repetitive exercises. 2014.

[7] Temiloluwa Prioleau, Elliot Moore, and Maysam Ghovanloo. Unobtrusive and wearable systems for automatic dietary monitoring. 2020.

[8] Tobias Röddiger, Christopher Clarke, Paula Breitling anf Tim Schneegans, Haibin Zhao, Hans Gellersen, and Micheal Beigl. Sensing with earables: A systematic literature review and taxonomy of phenomena. 2022.

[9] Nur Asmiza Selamat and Sawal hamid Md. Aki. Automatic food intake monitoring based on chewing activity: A survey. 2020.

Table 2: Performance of the algorithms with different train-test splits

peformance				
Algorithm	Train set	Test set	Accuracy	f1-score
Random forest	p1, p3 , p4	p2	0.65	0.60
Random forests	p1, p2	p4, p3	0.89	0.76
Random forest	p1	p2, p3, p4	0.74	0.33
Random forest	p2, p3, p4	p1	0.71	0.42
Random forest	p1, p2, p3, p4	p1, p2, p3, p4	0.72	0.58
Decision Tree	p1, p3 , p4	p2	0.64	0.59
Decision Tree	p1, p2	p4, p3	0.65	0.59
Decision Tree	p1	p2, p3, p4	0.65	0.27
Decision Tree	p2, p3, p4	p1	0.61	0.28
Decision Tree	p1, p2, p3, p4	p1, p2, p3, p4	0.67	0.48
Log. Reg.	p1, p3 , p4	p2	0.68	0.5
Log. Reg.	p1, p2	p4, p3	0.87	0.78
Log. Reg.	p1	p2, p3, p4	0.74	0.43
Log. Reg.	p2, p3, p4	p1	0.75	0.63
Log. Reg.	p1, p2, p3, p4	p1, p2, p3, p4	0.69	0.57
Sup. Vec. Mach.	p1, p3 , p4	p2	0.53	0.49
Sup. Vec. Mach.	p1, p2	p4, p3	0.73	0.65
Sup. Vec. Mach.	p1	p2, p3, p4	0.57	0.34
Sup. Vec. Mach	p2, p3, p4	p1	0.58	0.55
Sup. Vec. Mach.	p1, p2, p3, p4	p1, p2, p3, p4	0.58	0.48
K-NN	p1, p3 , p4	p2	0.74	0.61
K-NN.	p1, p2	p4, p3	0.80	0.59
K-NN.	p1	p2, p3, p4	0.68	0.52
K-NN.	p2, p3, p4	p1	0.64	0.28
K-NN.	p1, p2, p3, p4	p1, p2, p3, p4	0.69	0.57

Table 3: Performance: all, no autocorrelation, no FFT, no autocorrelation and no FFT

peformance				
Algorithm	**All feat**	**No autocorrelation**	**No FFT**	**No autocorrelation and no FFT**
Random forest	accuracy=0.89 f1-score=0.75	accuracy=0.80 f1-score=0.61	accuracy=0.87 f1-score=0.73	accuracy=0.92 f1-score=0.81
Decision Tree	accuracy=0.80 f1-score=0.61	accuracy=0.65 f1-score=0.59	accuracy=0.65 f1-score=0.59	accuracy=0.65 f1-score=0.59
Log. Reg.	accuracy=0.87 f1-score=0.78	accuracy=0.89 f1-score=0.82	accuracy=0.89 f1-score=0.82	accuracy=82 f1-score=0.67
Sup. Vec. Mach.	accuracy=0.45 f1-score=0.38	accuracy=0.69 f1-score=0.62	accuracy=0.79 f1-score=0.67	accuracy=0.73 f1-score=0.65
K-NN	accuracy=0.80 f1-score=0.59	accuracy=0.80 f1-score=0.59	accuracy=0.79 f1-score=0.61	accuracy=0.79 f1-score=0.62

A Gridsearch parameters: for computing all features

Features that were used in gridsearch, per algorithm:

Random forest:

- n_estimators: [100, 20, 50] (removed 200 when doing sequential feature selection due to runnning time)
- criterion: ['entropy', 'gini'] (removed 'gini' when doing sequential feature selection due to runnning time)
- max_dept': [None, 1, 2, 3, 10, 20]
- min_samples_leaf: [1, 2, 3, 5]
- min_samples_leaf: [1, 2, 3]
- class_weight: ['balanced', None] (removed None when computing Seq. Feat. Selection due to runnning time)

Decision tree:

- criterion: ['gini', 'entropy']
- max_depth: [None, 1, 2, 3, 10, 20]
- min_samples leaf: [1, 2, 3, 5]
- min_samples leaf: [1, 2, 3]
- class_weight: ['balanced', None]

Logistic regression:

- penalty: ['l1', 'l2', 'elastic', None]
- C: [0.001, 0.009, 0.01, 0.09, 1, 5, 10, 25]
- class_weight = ['balanced', None]
- random_state: [0, None]
- solver: ['liblinear', 'lbfgs']

K-nearest neighbors:

- n_neighbours: [3, 5, 8, 10, 20]

Support vector machine:

- kernel: ['rbf', 'sigmoid', 'poly']
- C: [1, 2, 3, 300, 500]
- max_iter: [1000, 100000, -1]

See next page for chosen parameters per train-test set.

Chosen hyperparameters after GridSearch (scikit)			
Algorithm	**Train set**	**Test set**	**Parameters**
Random forest	p1, p3 , p4	p2	class_weight=balanced, criterion=entropy, max_depth=2, min_samples_leaf=2, n_estimators=100
Random forest	p1, p2	p4, p3	class_weight=balanced, criterion=entropy, max_depth=None, min_samples_leaf=2, n_estimators=100
Random forest	p1	p2, p3, p4	class_weight=None, criterion=entropy, max_depth=10, min_samples_leaf=2, n_estimators=50
Random forest	p2, p3, p4	p1	class_weight=balanced, criterion=entropy, max_depth=10, min_samples_leaf=3, n_estimators=100
Random forest	p1, p2, p3, p4	p1, p2, p3, p4	class_weight=None, criterion=gini, max_depth=None, min_samples_leaf=1, n_estimators=50
Decision Tree	p1, p3 , p4	p2	class_weight=None, criterion=entropy, max_depth=2, min_samples_leaf=2
Decision Tree	p1, p2	p4, p3	class_weight=balanced, criterion=entropy, max_depth=2, min_samples_leaf=1
Decision Tree	p1	p2, p3, p4	class_weight=balanced, criterion=entropy, max_depth=20, min_samples_leaf=3
Decision Tree	p2, p3, p4	p1	class_weight=balanced, criterion=entropy, max_depth=None, min_samples_leaf=2
Decision Tree	p1, p2, p3, p4	p1, p2, p3, p4	class_weight=None, criterion=gini, max_depth=10, min_samples_leaf=1
Log. Reg.	p1, p3 , p4	p2	C=0.01, class_weight=None, penalty=l1, random_state=0, solver=liblinear
Log. Reg.	p1, p2	p4, p3	C=0.01, class_weight=balanced, penalty=l2, random_state=0, solver=lbfgs
Log. Reg.	p1	p2, p3, p4	C=5, class_weight=balanced, penalty=l2, random_state=0, solver=lbfgs
Log. Reg.	p2, p3, p4	p1	C=0.01, class_weight=None, penalty=l1, random_state=0, solver=liblinear
Log. Reg.	p1, p2, p3, p4	p1, p2, p3, p4	C=25, class_weight=balanced, penalty=l1, random_state=None, solver=liblinear
Sup. Vec. Mach.	p1, p3 , p4	p2	C=1, kernel=linear, max_iter=100000
Sup. Vec. Mach.	p1, p2	p4, p3	C=1, kernel=linear, max_iter=100000
Sup. Vec. Mach.	p1	p2, p3, p4	C=1, kernel=linear, max_iter=1000
Sup. Vec. Mach	p2, p3, p4	p1	C=1, kernel=linear, max_iter=100000
Sup. Vec. Mach.	p1, p2, p3, p4	p1, p2, p3, p4	C=1, kernel=linear, max_iter=100000
K-NN	p1, p3 , p4	p2	n_neighbors=15
K-NN.	p1, p2	p4, p3	n_neighbors=3
K-NN.	p1	p2, p3, p4	n_neighbors=20
K-NN.	p2, p3, p4	p1	n_neighbors=3
K-NN.	p1, p2, p3, p4	p1, p2, p3, p4	n_neighbors=8

B Gridsearch parameters: all features, no autocorrelation, no FFT, no autocorrelation and no FFT

Features that were used in gridsearch, per algorithm:

Random forest:

- All: class_weight=None, criterion='entropy', max_depth=None, min_samples_leaf=2, n_estimators=50
- No autocorr.: class_weight='balanced', criterion='entropy', max_depth=10, min_samples_leaf=3, n_estimators=20
- No FFT: class_weight=None, criterion='entropy', max_depth=10, min_samples_leaf=2, n_estimators=50
- No autocorr and no FFT: class_weight=None, criterion='entropy', max_depth=None, min_samples_leaf=2, n_estimators=50

Decision tree:

- All: class_weight=None, criterion='entropy', max_depth=20, min_samples_leaf=3
- No autocorr.: class_weight='balanced', criterion='entropy', max_depth=2, min_samples_leaf=1
- No FFT: class_weight='balanced', criterion='entropy', max_depth=2, min_samples_leaf=1
- No autocorr and no FFT: class_weight='balanced', criterion='entropy', max_depth=2, min_samples_leaf=2

Logistic regression:

- All: C=0.01, class_weight='balanced', penalty=l2, random_state=0, solver=lbfgs
- No autocorr.: C=1, class_weight='balanced', penalty=l1, random_state=0, solver=liblinear
- No FFT: C=0.09, class_weight='balanced', penalty=l1, random_state=0, solver=liblinear
- No autocorr and no FFT: C=0.01, class_weight='balanced', penalty=l2, random_state=0, solver=lbfgs

K-nearest neighbors:

- All: n_neighbors=3
- No autocorr.: n_neighbors=3
- No FFT: n_neighbors=3
- No autocorr and no FFT: n_neighbors=3

Support vector machine:

- All: C=1, kernel=linear, max_iter: 100000
- No autocorr.: C=1, kernel=linear, max_iter: 100000
- No FFT: C=1, kernel=linear, max_iter: 1000
- No autocorr and no FFT: C=1, kernel=linear, max_iter: 100000

C RF performance with Sequential Feature Selection

RF - Performance after Sequential forward feature selection (SFS)							
→ Number of features	#1	#2	#3	#5	#10	#20	#30
↓ Features							
f1 - mean			1				
2 - stand. dev.					0		
f3 - variance				1	0, 1, 3, 4		
f4 - rand. mean sqr.		1	1	1	2		
f5 - nr zero-cross.					2		
f6 - var. zero-cross.	1				5		
f7 - peak freq (FFT)				4	0		
f8 - peak power (FFT)		2	2	2			
f9 - nr. auto-corr. peaks				2	2		
f10 - weak peaks (ac)							
f11 - prominent peaks (ac)							
f12- max autocorr. value (ac)							
f13 - first peak after zero-cross. (ac)							
Random forest • accuracy: • f1-score:	 0.73 0.64	 0.87 0.79	 0.85 0.73	 0.90 0.82	 0.94 0.89		

The number in the table refer to the axis of the IMU:

0 : x-axis accelerometer
1 : y-axis accelerometer
2 : z-axis accelerometer
3 : x-axis gyroscope
4 : y-axis gyroscope
5 : z-axis gyroscope

Random forest:

- Nr. of features 1: class_weight='balanced', criterion='entropy', max_depth=1, min_samples_leaf=1, n_estimators=20
- Nr. of features 2: class_weight='balanced', criterion='entropy', max_depth=3, min_samples_leaf=2, n_estimators=20
- Nr. of features 3:class_weight='balanced', criterion='entropy', max_depth=3, min_samples_leaf=3, n_estimators=100
- Nr. of features 4: class_weight='balanced, criterion='entropy', max_depth=10, min_samples_leaf=1, n_estimators=50
- Nr. of features 10: class_weight=balanced', criterion='entropy', max_depth=20, min_samples_leaf=2, n_estimators=100
- Nr. of features 20: not determined due to long runtime
- Nr. of features 30: not determined due to long runtime

D DT performance with Sequential Feature Selection

DT - Performance after Sequential forward feature selection (SFS)							
→ Number of features	#1	#2	#3	#5	#10	#20	#30
↓ Features							
f1 - mean			0	0	0	0	0, 4
2 - stand. dev.				0		4	0, 1
f3 - variance		4	4	0, 4	4	4	0, 4, 5
f4 - rand. mean sqr.	1	1	1	1	1, 2	1, 4	0, 1, 2, 3
f5 - nr zero-cross.							0, 2
f6 - var. zero-cross.							0, 1, 4, 5
f7 - peak freq (ffT)					0	0, 2, 3	0, 1, 3
f8 - peak power (ffT)					3		
f9 - nr. auto-corr. peaks					3	3	0, 3
f10 - weak peaks (ac)					0	0, 1, 2, 3, 4	0, 5
f11 - prominent peaks (ac)					0	1, 2, 3, 4	0, 1, 2
f12- max autocorr. value (ac)					4		
f13 - first peak after zero-cross. (ac)						4, 5	0, 1
Decision tree • accuracy: • f1-score:	0.75 0.65	0.76 0.67	0.76 0.67	0.76 0.67	0.89 0.80	0.75 0.65	0.89 0.80

The number in the table refer to the axis of the IMU:

0 : x-axis accelerometer
1 : y-axis accelerometer
2 : z-axis accelerometer
3 : x-axis gyroscope
4 : y-axis gyroscope
5 : z-axis gyroscope

Decision tree parameters:

- Nr. of features 1: class_weight='balanced', criterion='gini', max_depth=2, min_samples_leaf=1
- Nr. of features 2: class_weight='balanced', criterion='gini', max_depth=3, min_samples_leaf=1
- Nr. of features 3:class_weight='balanced', criterion='gini', max_depth=3, min_samples_leaf=1
- Nr. of features 4: class_weight='balanced, criterion='gini', max_depth=3, min_samples_leaf=1
- Nr. of features 10: class_weight=None', criterion='entropy', max_depth=10, min_samples_leaf=2
- Nr. of features 20: class_weight='balanced', criterion='gini', max_depth=2, min_samples_leaf=3
- Nr. of features 30: class_weight='balanced', criterion='gini', max_depth=20, min_samples_leaf=3

E LR performance with Sequential Feature Selection

Log. Reg. - Performance after Sequential forward feature selection (SFS)							
→ Number of features	#1	#2	#3	#5	#10	#20	#30
↓ Features							
f1 - mean					0	0	
2 - stand. dev.					2	1, 2	
f3 - variance						1, 2	
f4 - rand. mean sqr		1	1	1	1, 2	0, 1, 2	
f5 - nr zero-cross.			3	3	3	3	
f6 - var. zero-cross.	1	1	1	1	1	0, 1	
f7 - peak freq (ffT)				0	0	0	
f8 - peak power (ffT)						2	
f9 - nr. auto-corr. peaks							
f10 - weak peaks (ac)				0	0,1	0, 1, 4	
f11 - prominent peaks (ac)							
f12- max autocorr. value (ac)						0, 3	
f13 - first peak after zero-cross. (ac)					0	0, 2	
Logistic Regression • accuracy: • f1-score:	0.77 0.62	0.75 0.65	0.77 0.69	0.77 0.69	0.80 0.72	0.87 0.78	0. 0.

The number in the table refer to the axis of the IMU:

0 : x-axis accelerometer

1 : y-axis accelerometer

2 : z-axis accelerometer

3 : x-axis gyroscope

4 : y-axis gyroscope

5 : z-axis gyroscope

Logistic regression:

- Nr. of features 1: C=0.001, class_weight='balanced', penalty=l2, random_state=0, solver=liblinear
- Nr. of features 2:C=0.001, class_weight='balanced', penalty=l2, random_state=0, solver=lbfgs
- Nr. of features 3: C=0.001, class_weight='balanced', penalty=l2, random_state=0, solver=lbfgs
- Nr. of features 4: C=0.001, class_weight='balanced', penalty=l2, random_state=0, solver=lbfgs
- Nr. of features 10: C=0.009, class_weight='balanced', penalty=l2, random_state=0, solver=lbfgs
- Nr. of features 20:
- Nr. of features 30:

F KNN performance with Sequential Feature Selection

KNN - Performance after Sequential forward feature selection (SFS)							
→ Number of features	#1	#2	#3	#5	#10	#20	#30
↓ Features							
f1 - mean					4	4	4
f2 - stand. dev.	3	3	3	3, 5	3, 5	3, 5	3, 5
f3 - variance							
f4 - rand. mean sqr.		2	2	2	2, 3	2, 3	2, 3
f5 - nr zero-cross.			5	5	2, 5	2, 5	2, 5
f6 - var. zero-cross.					0	0	0, 3
f7 - peak freq (ffT)					0	0, 1, 2	0, 1, 2, 3
f8 - peak power (ffT)							
f9 - nr. auto-corr. peaks				2	2	2	2
f10 - weak peaks (ac)						0, 1	0, 1, 2, 3
f11 - prominent peaks (ac)						0, 1	0, 1, 2,3
f12- max autocorr. value (ac)						0, 1	0, 1, 2, 3
f13 - first peak after zero-cross. (ac)						0, 1	0, 1, 2, 3
K-nearest neighbor • accuracy: • f1-score:	0.77 0.61	0.83 0.0.7	0.89 0.79	0.92 0.84	0.97 0.94	0.97 0.94	0.97 0.94

The number in the table refer to the axis of the IMU:

0 : x-axis accelerometer
1 : y-axis accelerometer
2 : z-axis accelerometer
3 : x-axis gyroscope
4 : y-axis gyroscope
5 : z-axis gyroscope

K-nearest neighbor:

- Size 1: n_neighbors=20
- Size 2: n_neighbors=3
- Size 3:n_neighbors=3
- Size 4: n_neighbors=5
- Size 10: n_neighbors=5
- Size 20: n_neighbors=5
- Size 30: n_neighbors=5

G SVM performance with Sequential Feature Selection

SVM - Performance after Sequential forward feature selection (SFS)							
→ Number of features	#1	#2	#3	#5	#10	#20	#30
↓ Features							
f1 - mean							1
2 - stand. dev.				1	1	1	1, 3
f3 - variance		1	1	1	1	1	1, 2, 4
f4 - rand. mean sqr.					2	2	1, 2
f5 - nr zero-cross.							
f6 - var. zero-cross.			0	0	0	0, 4, 5	0, 4, 5
f7 - peak freq (ffT)				0	0	0, 1, 4	0, 1, 2, 4
f8 - peak power (ffT)							7
f9 - nr. auto-corr. peaks					5	5	5
f10 - weak peaks (ac)					4	1, 2, 4	0, 1, 2, 4
f11 - prominent peaks (ac)						3	2, 3, 5
f12- max autocorr. value (ac)	2	2	2	2	2	1, 2, 4	1, 2, 4
f13 - first peak after zero-cross. (ac)					1, 4	1, 3, 4	1, 3, 4
Suport Vector machine • accuracy: • f1-score:	 0.68 0.58	 0.69 0.62	 0.69 0.62	 0.69 0.62	 0.87 0.76	 0.86 0.78	 0.82 0.71

The number in the table refer to the axis of the IMU:

0 : x-axis accelerometer
1 : y-axis accelerometer
2 : z-axis accelerometer
3 : x-axis gyroscope
4 : y-axis gyroscope
5 : z-axis gyroscope

Support vector machine:

- Nr. of features 1: C=500, kernel=linear, max_iter: 1000
- Nr. of features 2: C=300, kernel=linear, max_iter: 100000
- Nr. of features 3: C=300, kernel=linear, max_iter: 100000
- Nr. of features 4: C=3, kernel=linear, max_iter: 100000
- Nr. of features 10: C=500, kernel=linear, max_iter: 100000
- Nr. of features 20: C=500, kernel=linear, max_iter: 100000
- Nr. of features 30: C=1, kernel=linear, max_iter: 100000

For gridsearch, the following three kernels were omitted ['rbf','sigmoid', 'poly'] due to long runtime. This seemed a sensible choice, since the previous gridsearch optimal parameters all contained the 'linear' kernel.