ACTIVITY RECOGNITION FROM A SINGLE CHEST-MOUNTED ACCELEROMETER

Introducing Dataset

- The dataset collects data from a wearable accelerometer mounted on the chest.
- Uncalibrated Accelerometer Data are collected from 15 participants performing 7 activities.
- The dataset is intended for Activity Recognition research purposes.
- https://archive.ics.uci.edu/ml/datasets/Activity+Recognition+from+Single+Chest-Mounted+Accelerometer

Relevant Information

3

- Sampling frequency of the accelerometer: 52 Hz
- Accelerometer Data are Uncalibrated
- Number of Participants: 15
- Number of Activities: 7
- Data Format: CSV

Dataset Information

4

- Data are separated by participant
- Each file contains the following information
  - sequential number, x acceleration, y acceleration, z acceleration, label
- Labels are codified by numbers
  1: Working at Computer
  2: Standing Up, Walking and Going up\down stairs
  3: Standing
  4: Walking
  5: Going Up\Down Stairs
  6: Walking and Talking with Someone
  7: Talking while Standing

No.	x	y	z	label
0	1502	2215	2153	1
1	1667	2072	2047	1
2	1611	1957	1906	1
3	1601	1939	1831	1
4	1643	1965	1879	1
...				
...				
162500	1930	2383	2074	7
162500	1929	2385	2076	7
162500	1926	2385	2078	7
162500	1922	2387	2078	0

Reference Papers

- Casale, P. Pujol, O. and Radeva, P.
  - Human activity recognition from accelerometer data using a wearable device, IbPRIA'11, 289-296, Springer-Verlag, 2011
  - https://www.researchgate.net/publication/221258784_Human_Activity_Recognition_from_Accelerometer_Data_Using_a_Wearable_Device

Step 1: Compute the magnitude

- Compute the magnitude of the acceleration

$$A_m = \sqrt{A_x^2 + A_y^2 + A_z^2}$$

- Each time series *Ai*, with *i* = {*x*, *y*, *z*, *m*} has been filtered with a digital filter in order to separate low frequencies components and high frequencies components

Exercise: Compute the magnitude

- Use 1.csv
- Generate four time series *Ai*, with $i = \{x, y, z, m\}$
  - *Remove the records with error labels.*

Step 2: Frequency analysis

- The cut-off frequency has been set to **1 *Hz***, arbitrarily.
- In this way, we obtain for each time series, three more time series A_{ij} with $j = \{b, dc, ac\}$,
  - *b* represents the time series without filtering
  - *dc* represents the time series resulting from a low pass filtering
  - ac represents the time series resulting from a high pass filtering

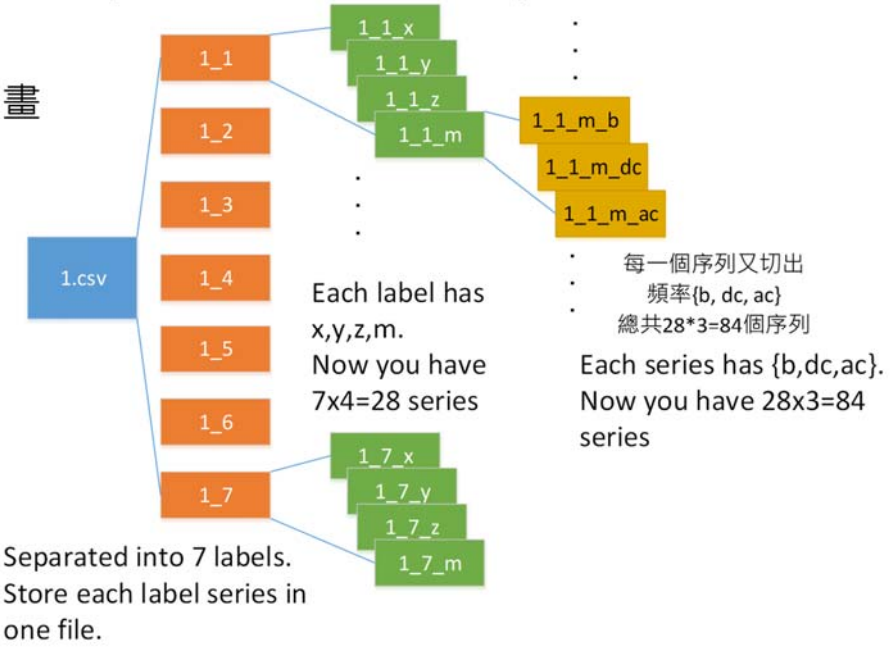
Exercise

- Read 1.csv and generate 84 time series A_{ijk}, with $i = \{1,2,\ldots,7\}$, $j = \{x, y, z, m\}$, $k = \{b, dc, ac\}$
- Plot these time series including b, dc and ac.
- Store each label series in one file. Thus, 7 labels should have 7 files. Each file has 12 columns {x, y, z, m} * {b, dc, ac}.

```
header = 'x_b, x_dc, x_ac, y_b, y_dc, y_ac,z_b, z_dc, z_ac,m_b, m_dc, m_ac'
np.savetxt('{}/label_{}.csv'.format(out_path, i), arr, delimiter=',', header=header, comments='')
```

```
df = pd.read_csv(f) # read csv file
print(df.columns)  # header
data = df.values   # to numpy array
```

- 每個label長度不一樣, 不要放在同一個array
- Plot series
  - 不同label分開畫



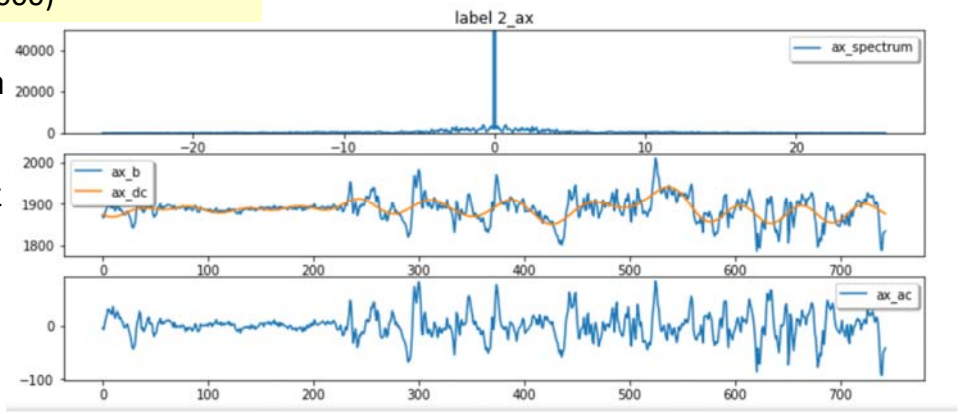
Plot Example


```
fig, (ax1, ax2, ax3) = plt.subplots(3,1)
fig.set_size_inches(12, 5)
ax1.set_ylim(0, 50000)
```

spectrum

b & dc

ac



PATHLIB MODULE

Pathlib module

- High-level path objects
  - https://docs.python.org/3/library/pathlib.html
  - This module offers classes representing filesystem paths with semantics appropriate for different operating systems.
  - Path classes are divided between pure paths, which provide purely computational operations without I/O, and concrete paths, which inherit from pure paths but also provide I/O operations.

Pathlib Example

- Find all sub directories' files that match the pattern L*.csv

```
import pathlib
import os
data_path = '/data/'
dpath = pathlib.Path(data_path)
sub_dir = [x for x in dpath.iterdir() if x.is_dir()]
for d in sub_dir:
    files = sorted(d.glob('L*.csv'))
    for f in files:
        print(f.name)
        os.remove(f)  # delete file
```

Pathlib Methods and properties

- Path.iterdir()
  - When the path points to a directory, yield path objects of the directory contents:

```
>>> p = Path('docs')
>>> for child in p.iterdir(): child
...
PosixPath('docs/conf.py')
PosixPath('docs/_templates')
PosixPath('docs/make.bat')
PosixPath('docs/index.rst'
```

Pathlib Methods and properties

- Path.is_dir()
  - Return True if the path points to a directory (or a symbolic link pointing to a directory), False if it points to another kind of file.
  - False is also returned if the path doesn't exist or is a broken symlink; other errors (such as permission errors) are propagated.
- Path.is_file()
  - Return True if the path points to a regular file (or a symbolic link pointing to a regular file), False if it points to another kind of file.
  - False is also returned if the path doesn't exist or is a broken symlink; other errors (such as permission errors) are propagated.

Pathlib Methods and properties

- Path.glob(pattern)
  - Glob the given relative pattern in the directory represented by this path, yielding all matching files (of any kind):

```
>>> sorted(Path('.').glob('*.py'))
[PosixPath('pathlib.py'), PosixPath('setup.py')]
```

  - The "**" pattern means "this directory and all subdirectories, recursively". In other words, it enables recursive globbing:

```
>>> sorted(Path('.').glob('**/*.py'))
[PosixPath('build/lib/pathlib.py'), PosixPath('docs/conf.py'),
 PosixPath('pathlib.py'), PosixPath('setup.py'),
PosixPath('test_pathlib.py')]
```

Pathlib Methods and properties

- PurePath.match(pattern)
  - Match this path against the provided glob-style pattern. Return True if matching is successful, False otherwise.
  - If pattern is relative, the path can be either relative or absolute, and matching is done from the right:

```
>>> PurePath('a/b.py').match('*.py')
True
>>> PurePath('/a/b/c.py').match('b/*.py')
True
>>> PurePath('/a/b/c.py').match('a/*.py')
False
```

Pathlib Methods and properties

- PurePath.name
  - A string representing the final path component, excluding the drive and root, if any:

```
>>> PurePosixPath('my/library/setup.py').name
'setup.py'
```

- PurePath.stem
  - The final path component, without its suffix:

```
>>> PurePosixPath('my/library.tar.gz').stem
'library.tar'
>>> PurePosixPath('my/library.tar').stem
'library'
```

Pathlib Example

- Check whether a path exists. If not, create one.

```
out_path = 'e:/tmp/'
outpath = pathlib.Path(out_path)

# combine and create a child path obj 'e:/tmp/outavi'
outavi_path = outpath.joinpath('outavi')

# check whether 'e:/tmp/outavi' exists
if not outavi_path.exists():
    outavi_path.mkdir()  # make directory
```

Pathlib Methods and properties

- PurePath.joinpath(*other)
  - Calling this method is equivalent to combining the path with each of the other arguments in turn:

```
>>> PurePosixPath('/etc').joinpath('passwd')
PurePosixPath('/etc/passwd')
>>> PurePosixPath('/etc').joinpath('init.d', 'apache2')
PurePosixPath('/etc/init.d/apache2')
```

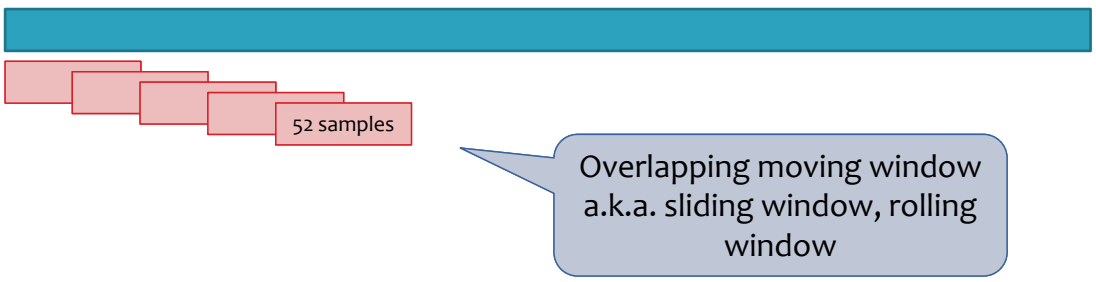
Pathlib Methods and properties

- Path.exists()
  - Whether the path points to an existing file or directory.
- Path.mkdir(mode=0o777, parents=False, exist_ok=False)
  - Create a new directory at this given path.
  - If parents is True, any missing parents of this path are created as needed.
  - If parents is False (the default), a missing parent raises FileNotFoundError.
  - If exist_ok is false (the default), FileExistsError is raised if the target directory already exists.
  - If exist_ok is true, FileExistsError exceptions will be ignored.

FEATURE EXTRACTION

Step 3: Feature extraction

- We extract features from data using windows of 52 samples, corresponding to 1 second of accelerometer data, with 50% of overlapping between windows.



Step 3: Feature extraction

- The paper calculates 319 features for each window including
  - **mean value (np.mean), standard deviation (np.std)**
  - Skewness (scipy.stats.skew), kurtosis (scipy.stats.kurtosis)
  - **Mean value of MinMax sum**
  - **Root mean squared value of integration of acceleration in a window**
  - correlation between each pairwise of accelerometer axis (not including magnitude)
  - energy of coefficients of seven level wavelet decomposition.

Important Features

Feature	Importance	Feature	Importance
Mean Value A_{zdc}	4.64	Mean Value A_{ydc}	3.86
MinMax A_{zdc}	4.61	Rms Velocity A_{ydc}	3.67
RMS Velocity A_{zdc}	4.23	Mean Value A_{zb}	3.59
RMS Velocity A_{mdc}	4.2	Mean Value A_{xdc}	3.57
RMS Velocity A_{xac}	4.14	MinMax A_{xdc}	3.52
Mean Value A_{mdc}	4.07	MinMax A_{zb}	3.51
MinMax A_{ydc}	3.92	Mean Value A_{yb}	3.33
Standard Deviation A_{xb}	3.9	Rms Velocity A_{xdc}	3.22
MinMax A_{mdc}	3.89	Rms Velocity A_{zb}	3.2
Standard Deviation A_{xdc}	3.87	MinMax A_{yb}	2.96

Overlapping moving window

- skimage.util.**view_as_windows**(arr_in, window_shape, step=1)
  - Windows are overlapping views of the input array, with adjacent windows shifted by a single row or column (or an index of a higher dimension).
- Parameters
  - arr_in: ndarray. N-d input array. Input should be a **contiguous array**.
  - window_shape: integer or tuple of length arr_in.ndim
    - Defines the shape of the elementary n-dimensional hyper-rectangle of the rolling window view.
  - Step: integer or tuple of length arr_in.ndim
    - Indicates step size at which extraction shall be performed. If integer is given, then the step is uniform in all dimensions.

skimage.util.view_as_windows()

- RuntimeWarning: Cannot provide views on a non-contiguous input array without copying.

```
print(np.info(a))
```

```
class:  ndarray
shape:  (26943,)
strides:  (96,)
itemsize:  8
aligned:  True
contiguous:  False
...
```

```
a1 = a.copy()
print(np.info(a1))
```

```
class:  ndarray
shape:  (26943,)
strides:  (8,)
itemsize:  8
aligned:  True
contiguous:  True
...
```

```
import numpy as np
from skimage.util.shape import view_as_windows
a = np.arange(16)
print(a.shape)
print(a)
b = view_as_windows(a, (4,), 2)
print(b.shape)
print(b)
win_mean = np.mean(b, axis = 1)
print(win_mean)
```

Axis 1: running horizontally across columns

```
(16,)
[ 0  1  2  3  4  5  6  7  8  9 10 11 12
13 14 15]
(7, 4)
[[ 0  1  2  3]
 [ 2  3  4  5]
 [ 4  5  6  7]
 [ 6  7  8  9]
 [ 8  9 10 11]
 [10 11 12 13]
 [12 13 14 15]]
[ 1.5  3.5  5.5  7.5  9.5 11.5 13.5]
```

```
a = np.arange(5*4).reshape(5, 4)
window_shape = (4, 3)
b = view_as_windows(a, window_shape)
```

print(b.shape)
print(b)

```
(2, 2, 4, 3)
[[[[ 0  1  2]
   [ 4  5  6]
   [ 8  9 10]
   [12 13 14]]

  [[ 1  2  3]
   [ 5  6  7]
   [ 9 10 11]
   [13 14 15]]]


 [[[ 4  5  6]
   [ 8  9 10]
   [12 13 14]
   [16 17 18]]

  [[ 5  6  7]
   [ 9 10 11]
   [13 14 15]
   [17 18 19]]]]
```

print(a)

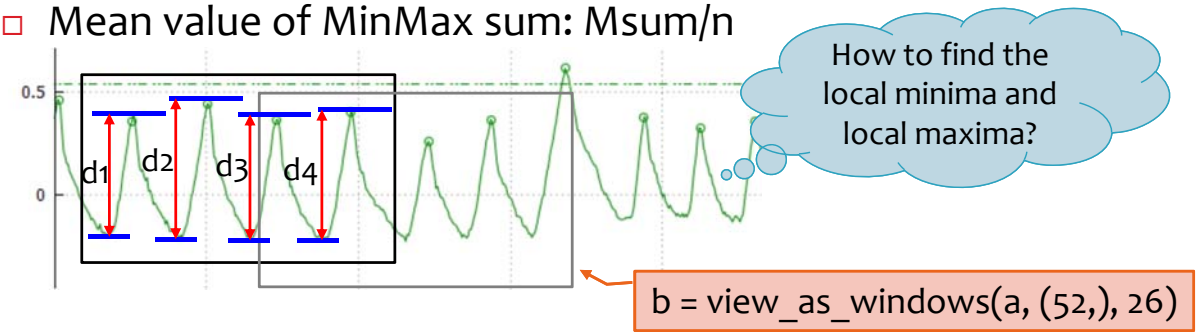
```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

print(b[0,1])

```
[[ 1  2  3]
 [ 5  6  7]
 [ 9 10 11]
 [13 14 15]]
```

Feature: Mean value of MinMax sum

- The Minmax sums are computed as the sum of all the differences of the ordered pairs of the peaks of the time series. $Msum = \sum_{i=1}^{n} d_i$
- Mean value of MinMax sum: Msum/n



Find local minima/maxima

- There are four functions in numpy to do this
  - scipy.signal.argrelextrema
  - scipy.signal.argrelmin (based on argrelextrema)
  - scipy.signal.argrelmax (based on argrelextrema)
  - scipy.signal.find_peaks
    - Powerful, more options

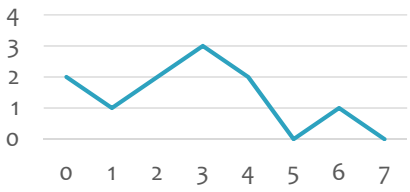
scipy.signal.argrelextrema

33

- scipy.signal.argrelextrema(data, comparator, axis=0, order=1, mode='clip')
  - Calculate the relative extrema of *data*.

```
from scipy.signal import argrelextrema
x = np.array([2, 1, 2, 3, 2, 0, 1, 0])
maxidx = argrelextrema(x, np.greater)
minidx = argrelextrema(x, np.less)
print(maxidx)
print(minidx)
```

```
(array([3, 6], dtype=int64),)
(array([1, 5], dtype=int64),)
```



scipy.signal.argrelmin and argrelmax

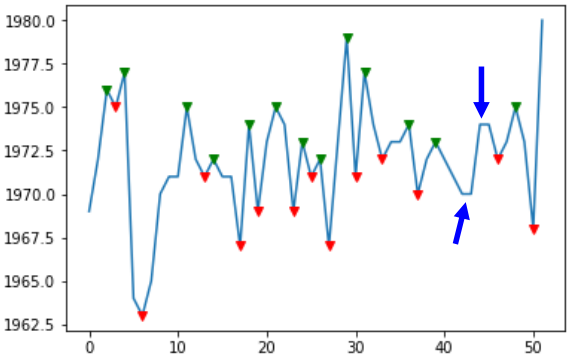
34

- scipy.signal.argrelmin (data, axis=0, order=1, mode='clip')
  - This function uses argrelextrema with np.less as comparator.
- scipy.signal.argrelmax(data, axis=0, order=1, mode='clip')
  - This function uses argrelextrema with np.greater as comparator.
- Parameter:
  - order: int, optional. How many points on each side to use for the comparison to consider comparator(n, n+x) to be True.

cannot detect flat extrema

35

- argrelextrema, argrelmin, and argrelmax **cannot** detect **flat** (more than one sample wide) minima and maxima



scipy.signal.find_peaks

36

- find_peaks(x, height=None, threshold=None, distance=None, prominence=None, width=None, wlen=None, rel_height=0.5, plateau_size=None)
  - This function takes a one-dimensional array and finds all local maxima by simple comparison of neighbouring values.
  - find_peaks can detect all local maxima, including flat ones.
  - Detect minima by calling it with negated data.
  - It's important to understand well its parameters width, threshold, distance and above all prominence to get a good peak extraction.

scipy.signal.find_peaks

- Parameters
  - height : number or ndarray or sequence, optional
    - Required height of peaks. Either a number, None, an array matching x or a 2-element sequence of the former. The first element is always interpreted as the minimal and the second, if supplied, as the maximal required height.
  - threshold : number or ndarray or sequence, optional
    - Required threshold of peaks, the vertical distance to its neighbouring samples. Either a number, None, an array matching x or a 2-element sequence of the former. The first element is always interpreted as the minimal and the second, if supplied, as the maximal required threshold.
  - distance : number, optional
    - Required minimal horizontal distance (>= 1) in samples between neighbouring peaks.

scipy.signal.find_peaks

- Parameters
  - prominence (突出): number or ndarray or sequence, optional
    - Required prominence of peaks. Either a number, None, an array matching x or a 2-element sequence of the former. The first element is always interpreted as the minimal and the second, if supplied, as the maximal required prominence.
  - width : number or ndarray or sequence, optional
    - Required width of peaks in samples. Either a number, None, an array matching x or a 2-element sequence of the former. The first element is always interpreted as the minimal and the second, if supplied, as the maximal required prominence.
  - wlen : int, optional
    - Used for calculation of the peaks prominences, thus it is only used if one of the arguments prominence or width is given. See argument wlen in peak_prominences for a full description of its effects.

scipy.signal.find_peaks

- Returns
  - **peaks** : ndarray. Indices of peaks in *x* that satisfy all given conditions.
  - **properties** : dict. A dictionary containing properties of the returned peaks:
    - peak_heights, left_thresholds, right_thresholds, prominences, right_bases, left_bases, width_heights, left_ips, right_ips, plateau_sizes, left_edges, right_edges

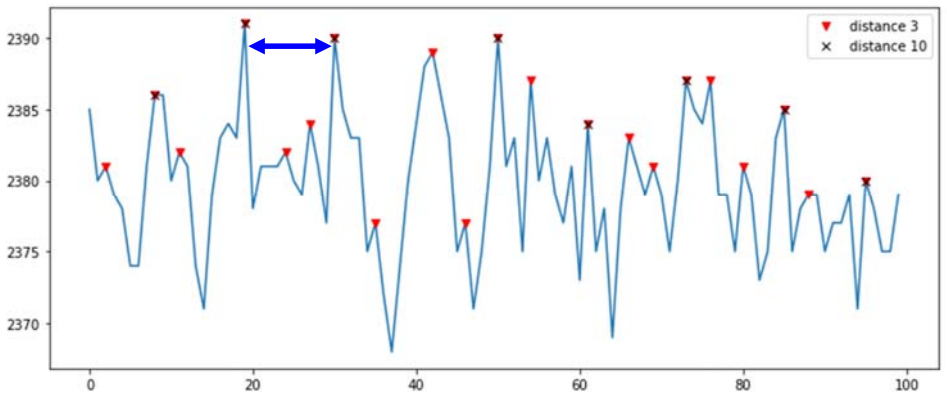
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
%matplotlib inline
data = np.genfromtxt('1.csv', delimiter=',',
usecols=(1,2,3,4),dtype='int32')
y = data[1000:1100,1]

peaks_d10, _ = find_peaks(y, distance=10)
peaks_d3, _ = find_peaks(y, distance=3)
print('distance 10 idx:',peaks_d10)
print('distance 3 idx:',peaks_d3)

plt.figure(figsize=(12, 5))
plt.plot(y)
plt.plot(peaks_d3, y[peaks_d3], "vr", label='distance 3')
plt.plot(peaks_d10, y[peaks_d10], "xk", label='distance 10')
plt.legend()
```

Parameters: distance

minimal horizontal distance between neighbouring peaks

Parameters: distance

- distance 10 idx: [8 19 30 50 61 73 85 95]
- distance 3 idx: [2 8 11 19 24 27 30 35 42 46 50 54 61 66 69 73 76 80 85 88 95]



Parameters: threshold

- Vertical distance to its neighbouring samples

```
peaks_d10, _ = find_peaks(y, threshold=5)
peaks_d3, _ = find_peaks(y, threshold=3)
print('threshold 5 idx:',peaks_d10)
print('threshold 3 idx:',peaks_d3)
print(y)
plt.figure(figsize=(12, 5))
plt.plot(y)
plt.plot(peaks_d3, y[peaks_d3], "vr", label='threshold 3')
plt.plot(peaks_d10, y[peaks_d10], "xk", label='distance 10')
plt.legend()
```

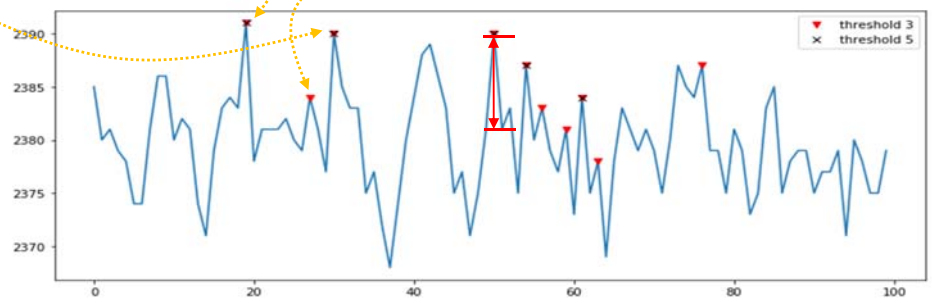
Parameters: threshold

- threshold 5 idx: [19 30 50 54 61]
- threshold 3 idx: [19 27 30 50 54 56 59 61 63 76]
- [2385 2380 2381 2379 2378 … 2381 2386 2386 2380 2382 2381 2374 2371
  2379 2383 2384 2383 **2391** 2378 2381 2381 2381 2382 2380 2379 **2384** 2381 2377
  2390 2385 2383 2383 2375

Idx 19

Idx 27

Idx 30



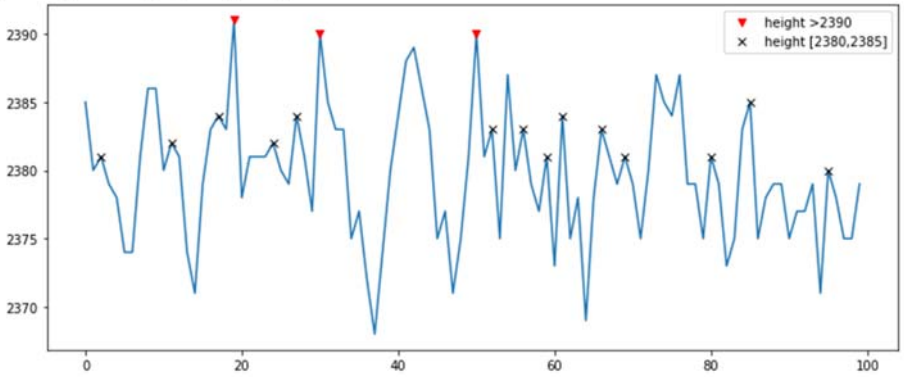
Parameters: height

- height of peaks

```
peaks_h2390, _ = find_peaks(y, height=2390)
peaks_h2380, _ = find_peaks(y, height=[2380,2385])
print('height >2390 :',y[peaks_h2390])
print('height [2380,2385] :',y[peaks_h2380])

plt.figure(figsize=(12, 5))
plt.plot(y)
plt.plot(peaks_h2390, y[peaks_h2390], "vr", label='height >2390')
plt.plot(peaks_h2380, y[peaks_h2380], "xk", label='height [2380,2385]')
plt.legend()
```

Parameters: height

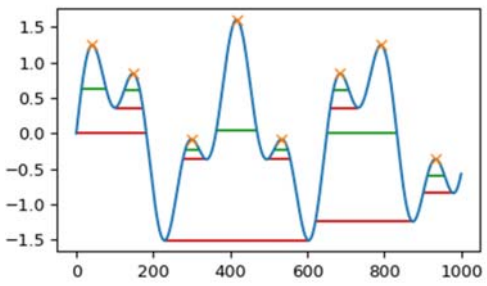
- height >2390 : [2391 2390 2390]
- height [2380,2385] : [2381 2382 2384 2382 2384 2383 2383 2381 2384 2383 2381 2381 2385 2380]



peak_widths()

- scipy.signal.peak_widths(*x*, *peaks*, *rel_height=0.5*, *prominence_data=None*, *wlen=None*)
  - This function calculates the width of a peak in samples at a relative distance to the peak's height and prominence.

- Green: calculate their widths at the relative height of 0.5 (contour line at half the prominence height)
- Red: calculate their widths at the relative height of 1 (at the lowest contour line at full prominence height).



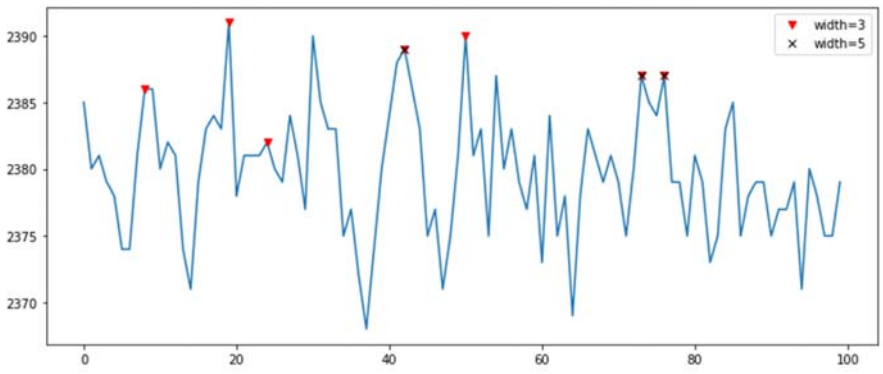
Parameters: width

- width of peaks in samples

```
peaks_w3, _ = find_peaks(y, width=3)
peaks_w5, _ = find_peaks(y, width=5)
print('width=3 :',peaks_w3)
print('width=5 :',peaks_w5)
print(y)
plt.figure(figsize=(12, 5))
plt.plot(y)
plt.plot(peaks_w3, y[peaks_w3], "vr", label='width=3')
plt.plot(peaks_w5, y[peaks_w5], "xk", label='width=5')
plt.legend()
```

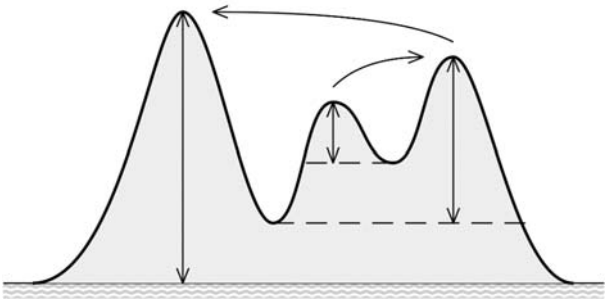
Parameters: width

- width=3 : [8 19 24 42 50 73 76]
- width=5 : [42 73 76]



Peak Prominences

- Vertical arrows show the topographic prominence (突出) of three peaks on an island.
  - The dashed horizontal lines show the lowest contours that do not encircle higher peaks. Curved arrows point from a peak to its parent.

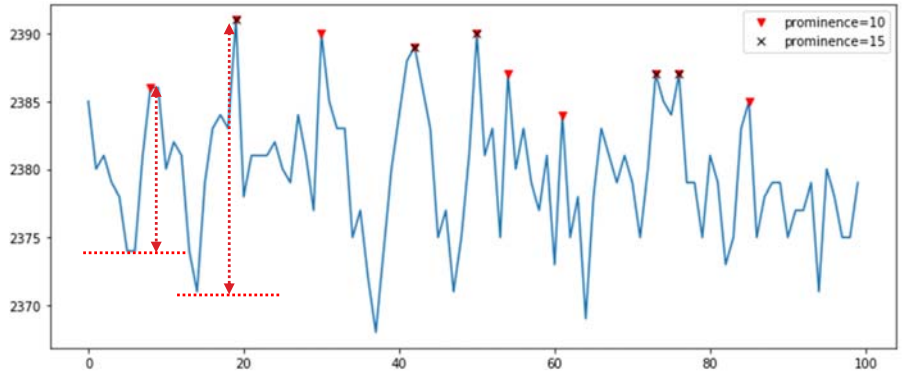


Parameters: prominence


```
peaks_p10, _ = find_peaks(y, prominence=10)
peaks_p15, _ = find_peaks(y, prominence=15)
print('prominence=10 :',peaks_p10)
print('prominence=15 :',peaks_p15)
print(y)
plt.figure(figsize=(12, 5))
plt.plot(y)
plt.plot(peaks_p10, y[peaks_p10], "vr", label='prominence=10')
plt.plot(peaks_p15, y[peaks_p15], "xk", label='prominence=15')
plt.legend()
```

Parameters: prominence

- prominence=10 : [8 19 30 42 50 54 61 73 76 85]
- prominence=15 : [19 42 50 73 76]

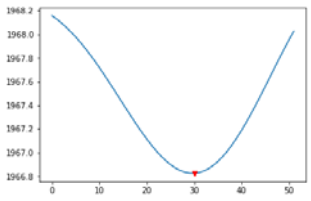


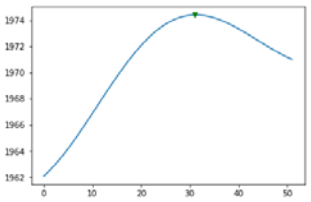
peak_prominences()

- scipy.signal.peak_prominences(*x*, *peaks*, *wlen=None*)
  - Calculate the prominence of each peak in a signal.
  - The prominence of a peak measures how much a peak stands out from the surrounding baseline of the signal and is defined as the vertical distance between the peak and its lowest contour line.

Feature: Mean value of MinMax sum

- If no minimum or maximum
  - Mean value of MinMax sum is set to **NaN**
    - **np.NaN**
  - Later, use pandas fillna() to fill the NaN
    - df.fillna(method='ffill', inplace = True)
    - df.fillna(method='bfill', inplace = True)





	mean_xb	std_xb	minmax_xb	rms_xb	mean_yb	std_yb	...
1			NaN				
...							

pandas.DataFrame.fillna

- DataFrame.fillna(*value=None, method=None, axis=None, inplace=False, limit=None, downcast=None, **kwargs*)
  - Fill NA/NaN values using the specified method.
- Parameters
  - **value** : *scalar, dict, Series, or DataFrame*
  - **method** : *{'backfill', 'bfill', 'pad', 'ffill', None}, default None*
  - **axis** : *{0 or 'index', 1 or 'columns'}*
  - **inplace** : *boolean, default False.* If True, fill in place.

Feature: RMS velocity

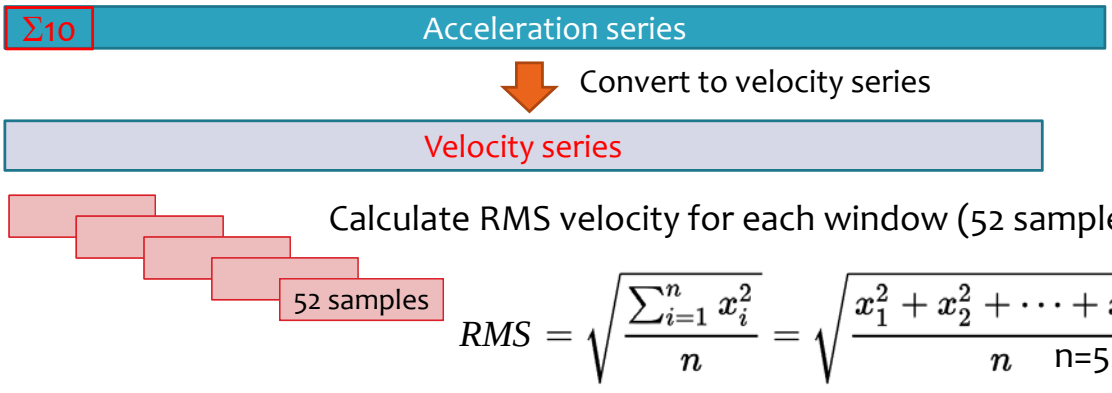
- Instantaneous acceleration $\mathbf{a} = \lim_{\Delta t \to 0} \frac{\Delta \mathbf{v}}{\Delta t} = \frac{d\mathbf{v}}{dt}$
- It can be seen that the integral of the acceleration function $a(t)$ is the velocity function $v(t)$

$$\mathbf{v} = \int \mathbf{a}\, dt \quad \text{(Continuous)} \qquad v = \sum_t a \quad \text{(Discrete)}$$

- Root mean squared value of integration of acceleration (velocity) in a window
  - The integral has been approximated using running sums with step equals to 10 samples.

Feature: RMS velocity

- RMS velocity




```
rms = np.sqrt(np.mean(np.square(b), axis = 1))
```

Feature vector

57

- Calculate the following 4 features for 84 time series A_{ijk}, with $i = \{1,2,\ldots,7\}$, $j = \{x, y, z, m\}$, $k = \{b, dc, ac\}$
  - mean value, standard deviation
  - Mean value of MinMax sum, RMS velocity
- Store one label features in one file
  - Each file has j x k x 4= **48 columns** (features)
  - The feature vector dimensionality is 48

mean_x_b, std_x_b, minmax_x_b, rms_x_b,mean_ x_dc, std_ x_dc, minmax_ x_dc, rms_ x_dc,mean_ x_ac, std_ x_ac, minmax_ x_ac, rms_ x_ac,mean_ y_b, std_ y_b, minmax_ y_b, rms_ y_b,mean_ y_dc, std_ y_dc, minmax_ y_dc, rms_ y_dc,mean_ y_ac, std_ y_ac, minmax_ y_ac, rms_ y_ac,mean_z_b, std_z_b, minmax_z_b, rms_z_b,mean_ z_dc, std_ z_dc, minmax_ z_dc, rms_ z_dc,mean_ z_ac, std_ z_ac, minmax_ z_ac, rms_ z_ac,mean_m_b, std_m_b, minmax_m_b, rms_m_b,mean_ m_dc, std_ m_dc, minmax_ m_dc, rms_ m_dc,mean_ m_ac, std_ m_ac, minmax_ m_ac, rms_ m_ac

Verify your results

58

- Feature_1.csv

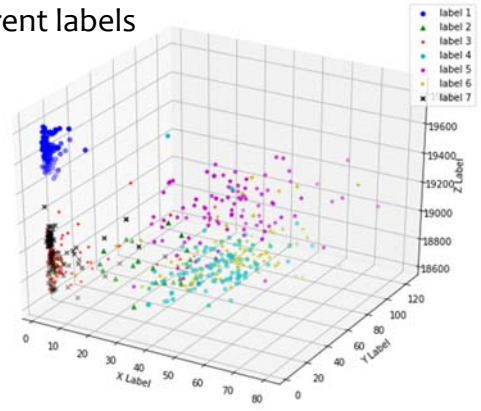
	mean_x_b	std_x_b	minmax_x_	rms_x_b	mean_ x_dc	std_ x_dc	minmax_ x_	rms_ x_dc	mean_ x_ac	std_ x_ac	minmax_ x_	rms_ x_ac
2	1971.37	3.16	5.13	19713.06	1971.08	0.40	0.72	19712.30	0.28	3.27	5.10	10.35
3	1971.96	3.28	5.07	19721.75	1972.00	1.05	2.81	19721.99	-0.03	3.20	4.82	8.23
4	1973.00	3.13	5.81	19731.48	1973.01	0.53	0.90	19730.93	-0.01	3.15	5.45	8.78
5	1973.12	3.10	5.05	19730.37	1973.13	0.26	0.63	19730.95	-0.02	3.12	5.07	8.69
6	1972.92	2.65	4.10	19725.35	1972.85	0.44	1.27	19727.99	0.07	2.68	4.11	10.93
7	1972.69	3.14	4.79	19727.60	1972.63	0.37	1.02	19726.26	0.06	3.09	4.79	10.31
8	1972.50	3.75	6.56	19727.02	1972.62	0.38	nan	19725.96	-0.12	3.70	5.84	6.78

- Feature_5.csv

	mean_x_b	std_x_b	minmax_x_	rms_x_b	mean_ x_dc	std_ x_dc	minmax_ x	rms_ x_dc	mean_ x_ac	std_ x_ac	minmax_ x	rms_ x_ac
2	1917.96	33.93	19.22	19127.57	1916.94	35.08	nan	19120.37	1.02	15.70	12.68	107.60
3	1886.81	39.82	29.33	18850.22	1885.65	26.87	nan	18838.81	1.16	23.76	18.20	154.35
4	1896.46	36.29	32.75	18974.24	1894.58	21.09	9.23	18992.31	1.89	24.68	29.13	169.65
5	1911.12	25.41	38.67	19110.65	1911.18	3.63	9.23	19098.91	-0.07	25.11	38.68	144.66
6	1892.48	44.70	46.30	18917.52	1893.16	22.55	60.52	18897.17	-0.68	39.91	48.97	206.28
7	1891.58	68.48	83.50	18916.71	1887.34	25.58	nan	18909.84	4.24	57.78	74.35	308.26

Data Visualization: mplot3d

- Mplot3d, http://matplotlib.org/examples/mplot3d/scatter3d_demo.html
- To see which features can be used to discriminate these activities
  - Use different colors and markers for different labels
  - Choose any three features and plot them.



Protocol

- Select 100 points

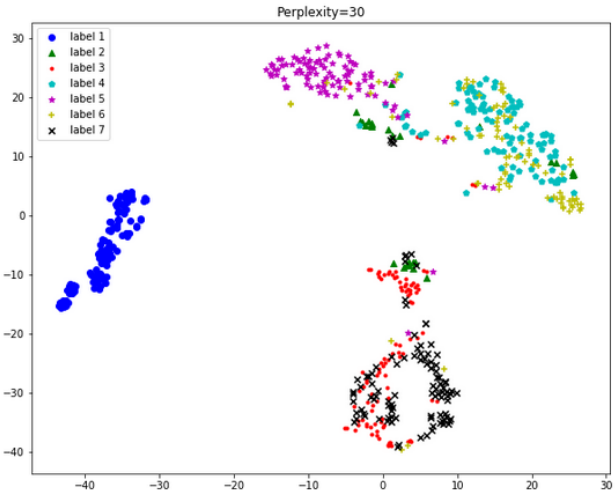
```
n = 100
np.random.seed(100)
idx = np.random.choice(data.shape[0], min(n, data.shape[0]), replace=False)
arr = data[idx,:]
color = ['b', 'g', 'r','c','m','y','k']
marker = ['o','^', '.','p','*','+','x']
x = 1 # dim 1, starting from 0
y = 2 # dim 2
z = 3 # dim 3
```

Data Visualization: t-SNE

- t-SNE (t-distributed Stochastic Neighbor Embedding)
  - t-SNE is a tool to visualize high-dimensional data.
  - It converts similarities between data points to joint probabilities and tries to minimize the KL divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.
  - It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high.
  - https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

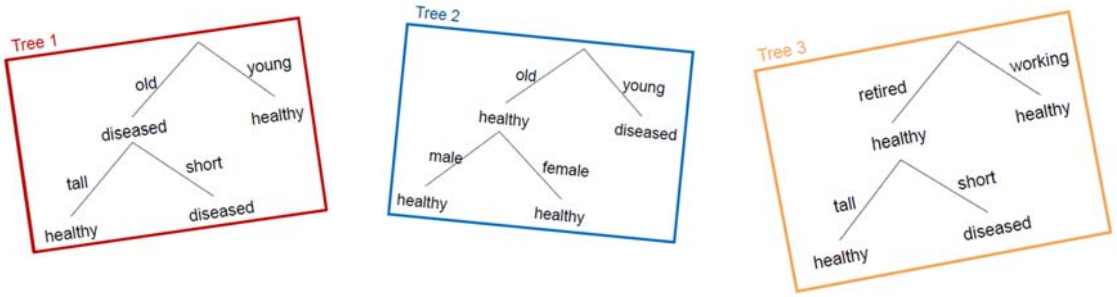
t-SNE Result

- perplexity = 30
- n_components = 2
- Use the same 100 samples



Random forest classifier

- Breiman, L.: Random Forests. Machine Learning 45(1), 5–32 (2001)
- A powerful new approach to data exploration, data analysis, and predictive modeling
- Random Forest is a collection of binary trees




```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pathlib
from sklearn.ensemble import RandomForestClassifier
n = 100  # select 100 training samples
np.random.seed(100) # fix the random number generator
fea_lst = []
y_lst = []
for i, f in enumerate(files):
    print(i, f.name)
    df = pd.read_csv(f)
    df.fillna(method='ffill', inplace = True)
    df.fillna(method='bfill', inplace = True)
    data = df.values
    idx = np.random.choice(data.shape[0], min(n, data.shape[0]),replace=False)
    arr = data[idx,:]
    fea_lst.append(arr)
    y_lst.append(np.full(arr.shape[0], i))
    print('df.shape {}, select {} features'.format(df.shape, arr.shape[0]))
```

```
fea_name = list(df.columns)
x_data = np.vstack(fea_lst)
y_label = np.hstack(y_lst)
print(y_label)
print('x_data:',x_data.shape)
print('y_label:',y_label.shape)

clf = RandomForestClassifier(n_estimators=100,  random_state=0)
clf.fit(x_data, y_label) #  Build a forest of trees from the training set (X, y)
print(clf.score(x_data, y_label)) #  Returns the mean accuracy
# print(clf.feature_importances_)
fea_dict= {}
for i, fea in enumerate(fea_name):
    fea_dict[fea] = clf.feature_importances_[i]

fea_import = [(v, k) for k, v in fea_dict.items() ]
fea_import = sorted(fea_import, reverse = True)
print(fea_import)
```

```
0 feature_1.csv
df.shape (1034, 48), select 100 features
1 feature_2.csv
df.shape (27, 48), select 27 features
2 feature_3.csv
df.shape (342, 48), select 100 features
3 feature_4.csv
df.shape (825, 48), select 100 features
4 feature_5.csv
df.shape (96, 48), select 96 features
5 feature_6.csv
df.shape (88, 48), select 88 features
6 feature_7.csv
df.shape (2575, 48), select 100 features
[0 0 0 ... 6 6 6]
x_data: (611, 48)
y_label: (611,)
mean accuracy: 1.0
```

```
(0.049568055101517065, 'mean_x_b'),
(0.04370705943602313, ' rms_ x_dc'),
(0.04158527009299771, 'mean_ x_dc'),
(0.0387946200649185, ' rms_z_b'),
(0.03850317617170872, ' rms_ m_ac'),
(0.037917685233579274, 'mean_ z_dc'),
(0.03679943959893353, ' rms_ z_dc'),
(0.03662608452791006, 'mean_z_b'),
(0.03487082364825953, 'mean_ m_dc'),
(0.03464635935090369, ' rms_x_b'),
(0.02922324634874958, ' rms_ m_dc'),
(0.027748125631727665, 'mean_m_b'),
(0.02711904464950775, ' rms_m_b'),
(0.026808337784851925, ' std_ m_ac'),
(0.026413551845069175, ' std_ y_b'),
(0.02427264741100405, ' rms_ z_ac'),
...
```

Midterm

- Reorganize your program about frequency analysis and feature extraction to process the whole 15-user dataset (1.csv, 2.csv, ..., 15.csv).
  - Put the code of frequency analysis and feature extraction into one jupyter notebook.
- Create directories for each user data file.
  - Create 15 directories: user1, user2, … ,user15
  - Each directory contains the label and feature data for the user.
- Use user1's features as training data to predict other users data, and show the `clf.score`
  - `Only show user1 to user8 because there are some problems in user9's features.`

protocol

- In user 1 training data
  - `n = 100  # select 100 training samples`
  - `np.random.seed(100) # fix the random number generator`
  - `clf = RandomForestClassifier(n_estimators=100, random_state=0)`
- In other users data
  - Use all features. Not just select 100 samples.

My Results

- predict, d:/tmp/user2
- mean accuracy: 0.2678022237993849
- predict, d:/tmp/user3
- mean accuracy: 0.021364795918367346
- predict, d:/tmp/user4
- mean accuracy: 0.2711502535361623
- predict, d:/tmp/user5
- mean accuracy: 0.5180281116316969
- predict, d:/tmp/user6
- mean accuracy: 0.220162224797219
- predict, d:/tmp/user7
- mean accuracy: 0.206675994403358
- predict, d:/tmp/user8
- mean accuracy: 0.1575958353052532
- predict, d:/tmp/user9
- ---
- ValueError Traceback (most recent call last)

Only show user1 to user8 because there are some problems in user9's features.