Mining Smartphone Data (with Python)

@neal_lathia

PyData London 2016


Information is Beautif
Artificial Intelligence
A Modern Approach
SECOND EDITION
Russell
Norvig
Predicting Structured Data
SAMSUNG
SAMSUNG
F9
F10
F11
F12
prt scr
sys rq
scroll lock

Smartphones have sensors!

- Accelerometer (acceleration)
- Gyroscope (orientation)
- GPS, Wi-Fi (location)
- ...

- Microphone (sound)
- Bluetooth (co-location)

Smartphones have sensors!

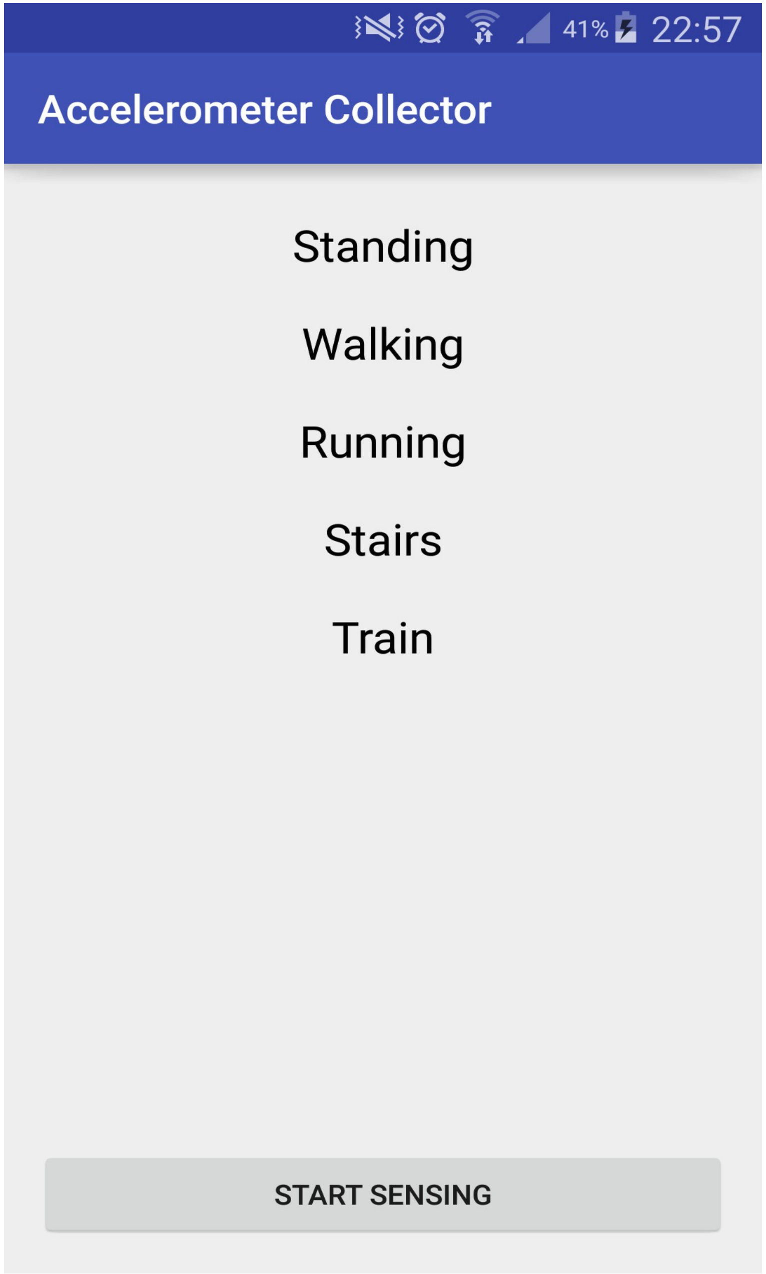
- Accelerometer (acceleration)
- Gyroscope (orientation)
- GPS, Wi-Fi (location)
- ...

- Microphone (sound)
- Bluetooth (co-location)

This talk

- Collecting accelerometer data
- A peek at the raw data
- Magnitude data
- Applications
- Feature extraction
- Focus on classification
- https://github.com/nlathia/pydata_2016

Collecting Data

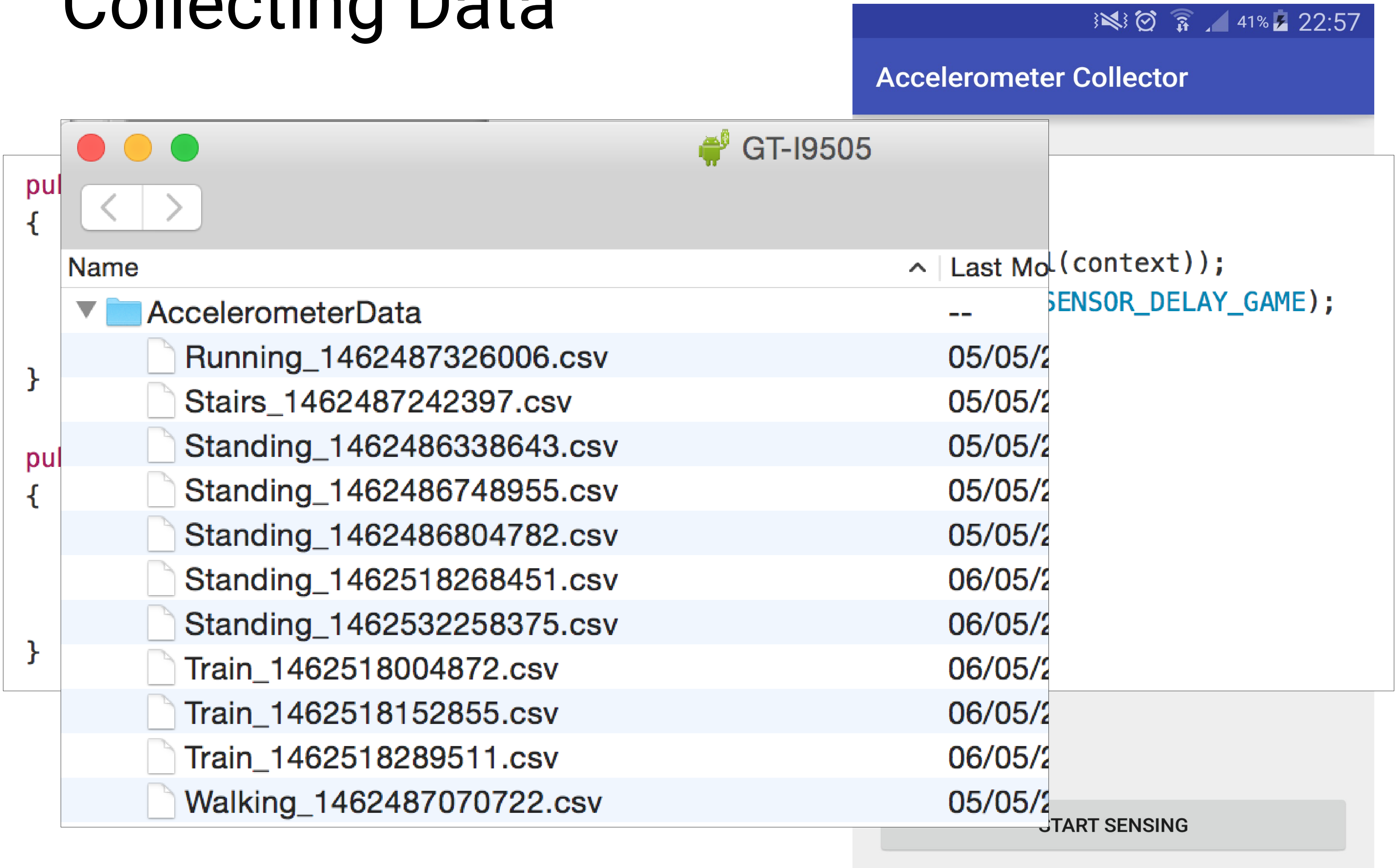


Collecting Data


```
public void start(final Context context) throws IOException
{
    fileWriter = new DataWriter(context, LabelPreferences.getLabel(context));
    mSensorManager.registerListener(this, mSensor, SensorManager.SENSOR_DELAY_GAME);
    isSensing = true;
}

public void stop() throws IOException
{
    fileWriter.finish();
    mSensorManager.unregisterListener(this);
    isSensing = false;
}
```

Collecting Data



The raw data

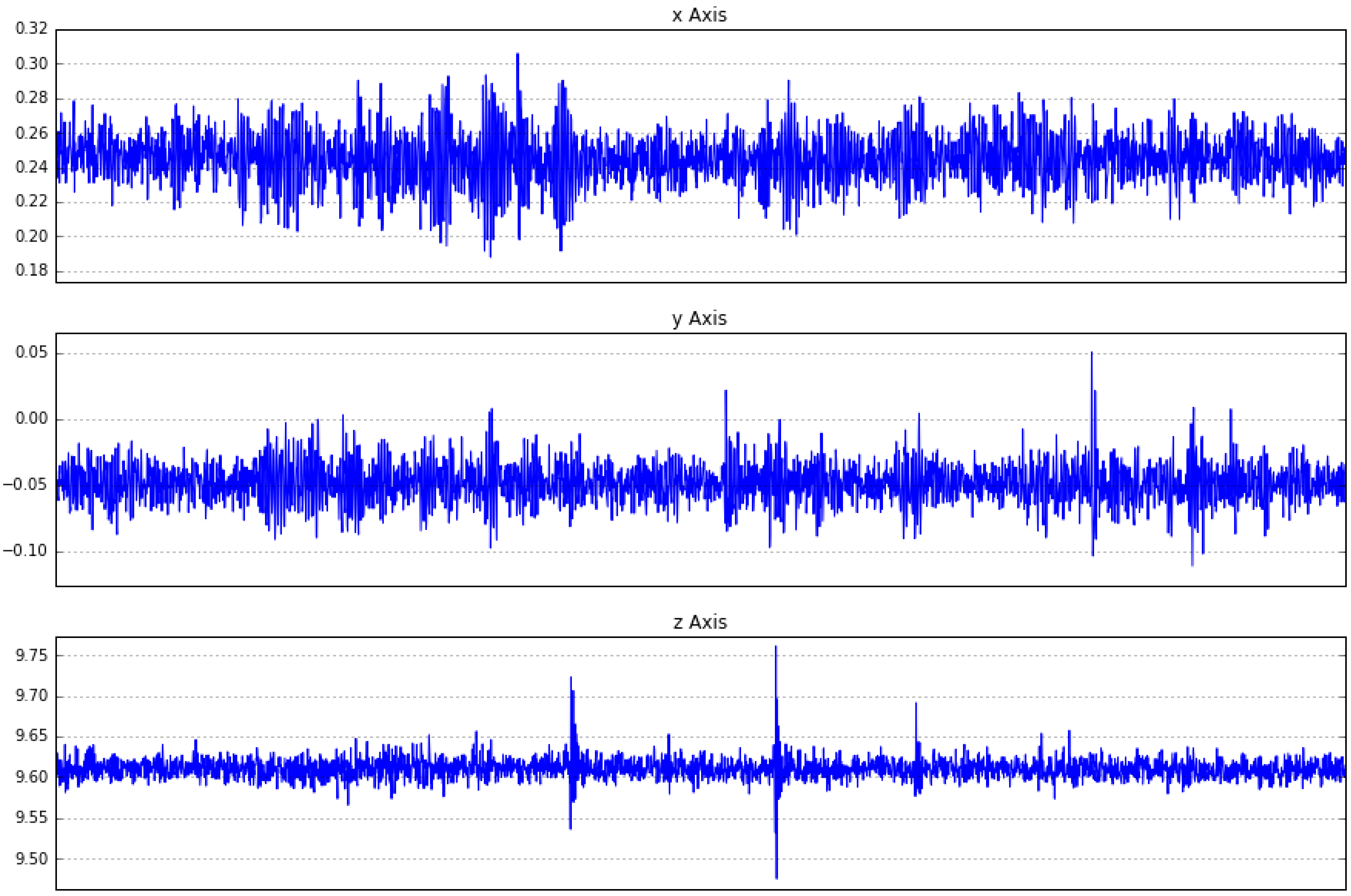
```
STANDING = pd.read_csv('../Data/Standing_1462486804782.csv', hea
WALKING = pd.read_csv('../Data/Walking_1462487070722.csv', heade
RUNNING = pd.read_csv('../Data/Running_1462487326006.csv', heade
STAIRS = pd.read_csv('../Data/Stairs_1462487242397.csv', header=
ON_TRAIN = pd.read_csv('../Data/Train_1462518004872.csv', header

STANDING.head()
```

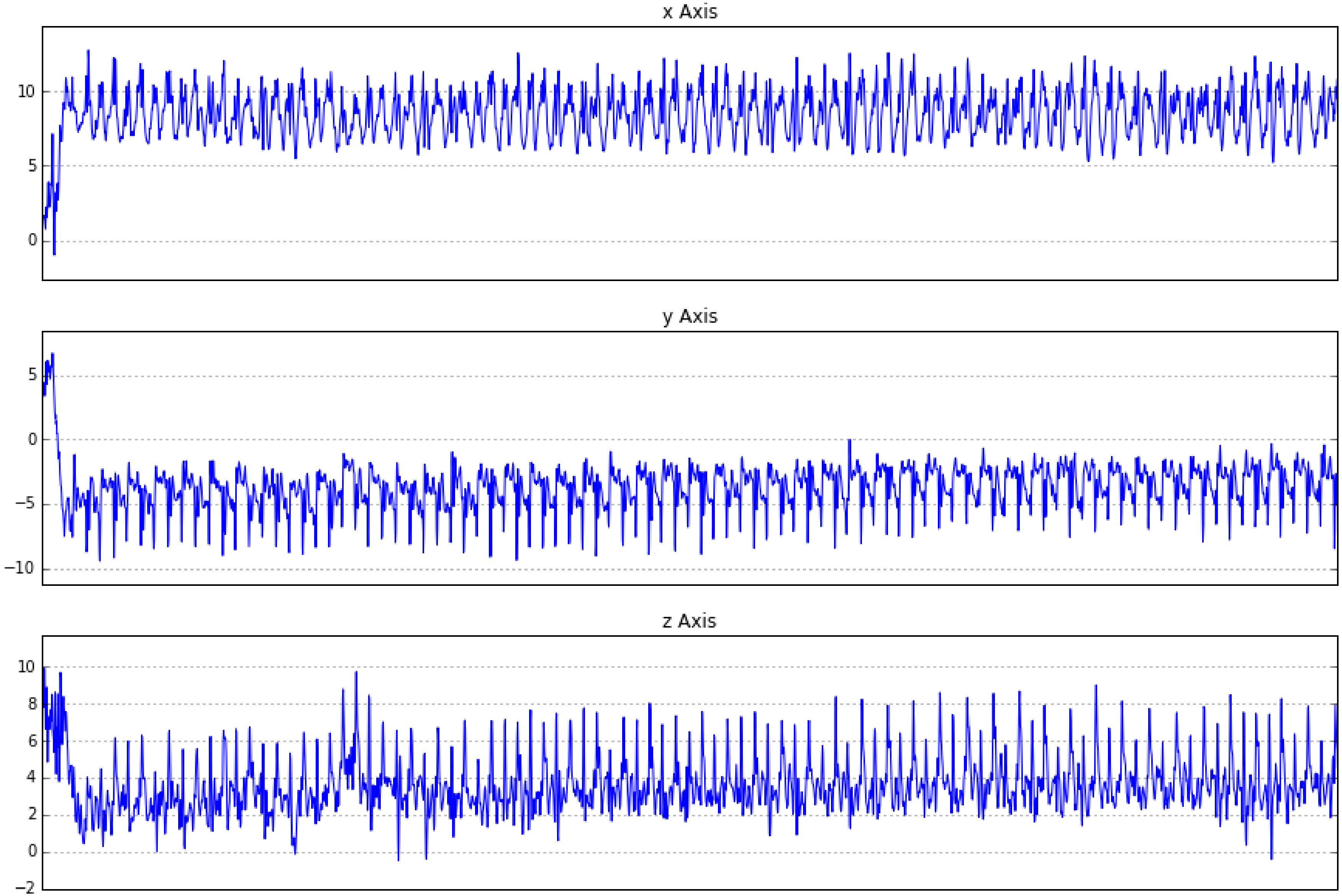
]:

	timestamp	xAxis	yAxis	zAxis
0	1462486804801	0.260968	-0.056862	9.611523
1	1462486804801	0.260968	-0.056862	9.611523
2	1462486804801	0.260968	-0.056862	9.611523
3	1462486804801	0.260968	-0.056862	9.611523
4	1462486804801	0.260968	-0.056862	9.611523

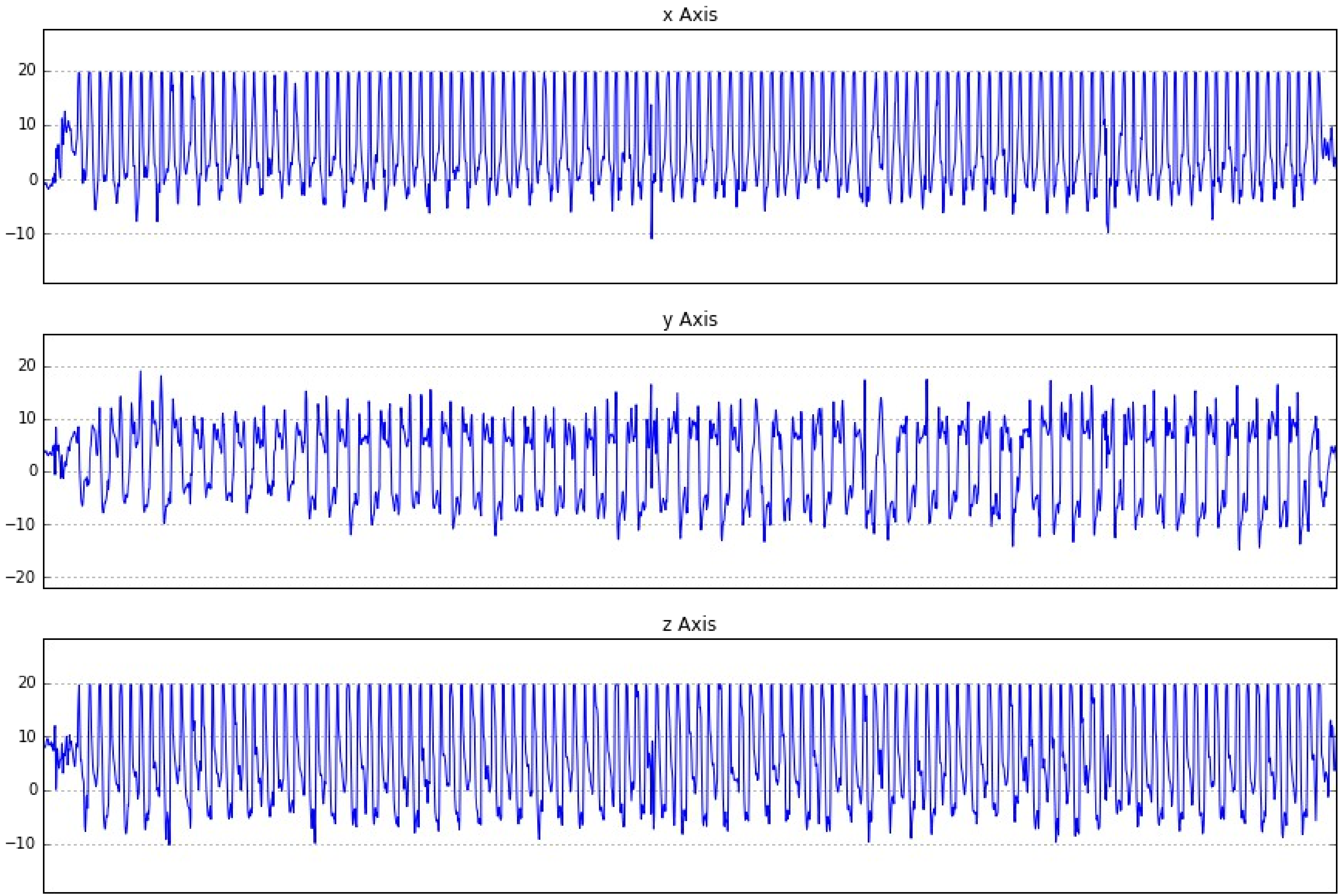
Raw: Standing



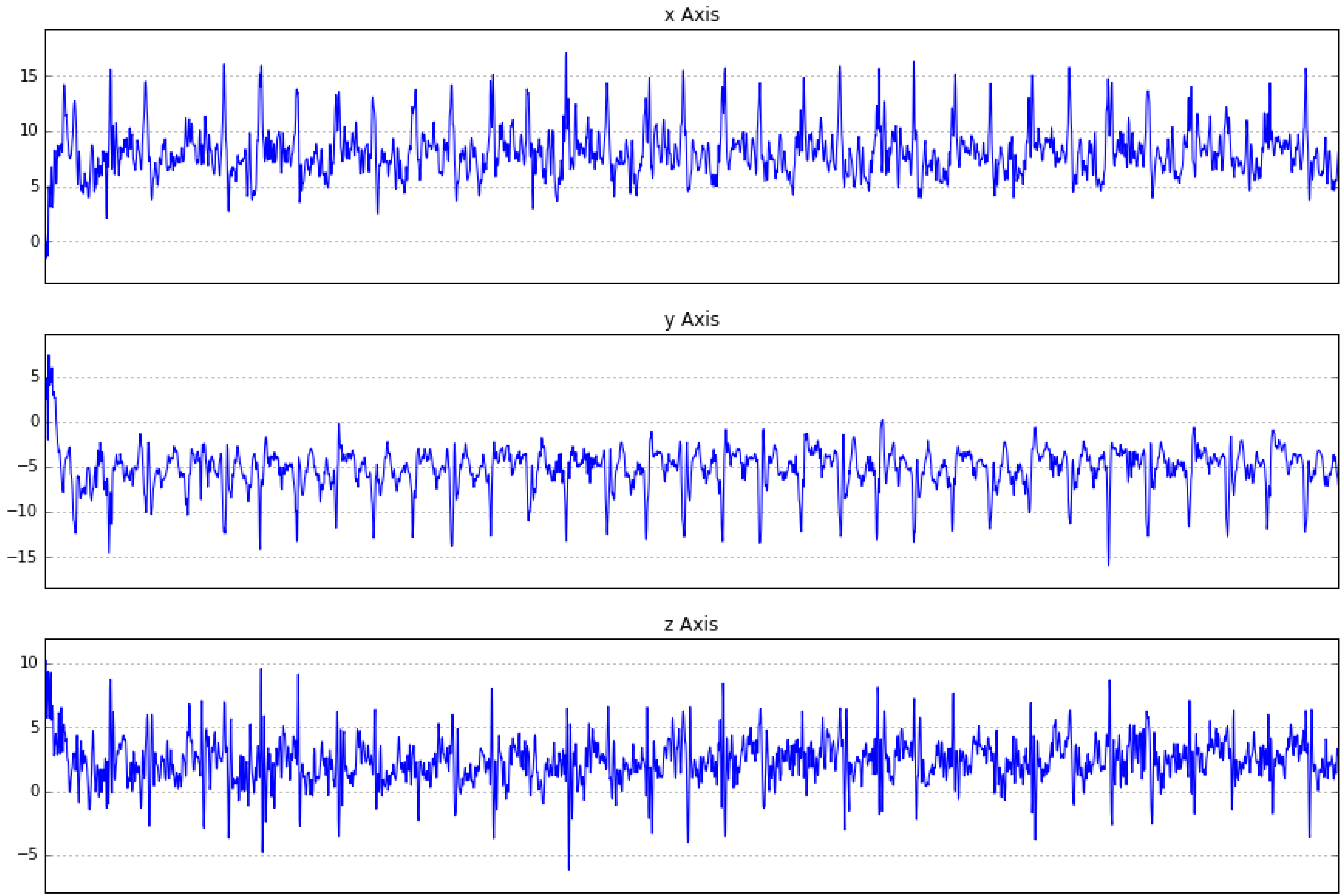
Raw: Walk



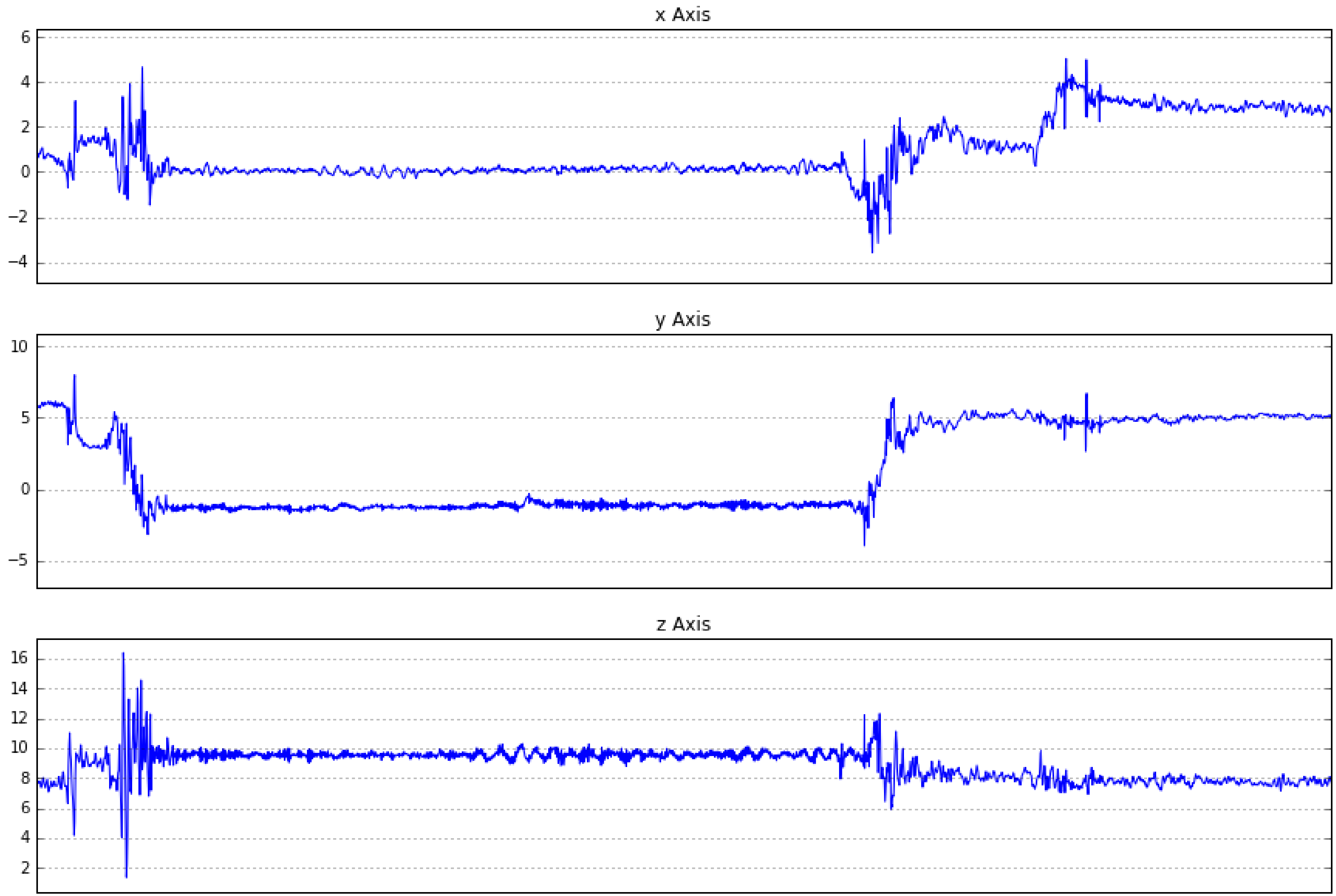
Raw: Run



Raw: Stairs



Raw: On a Train

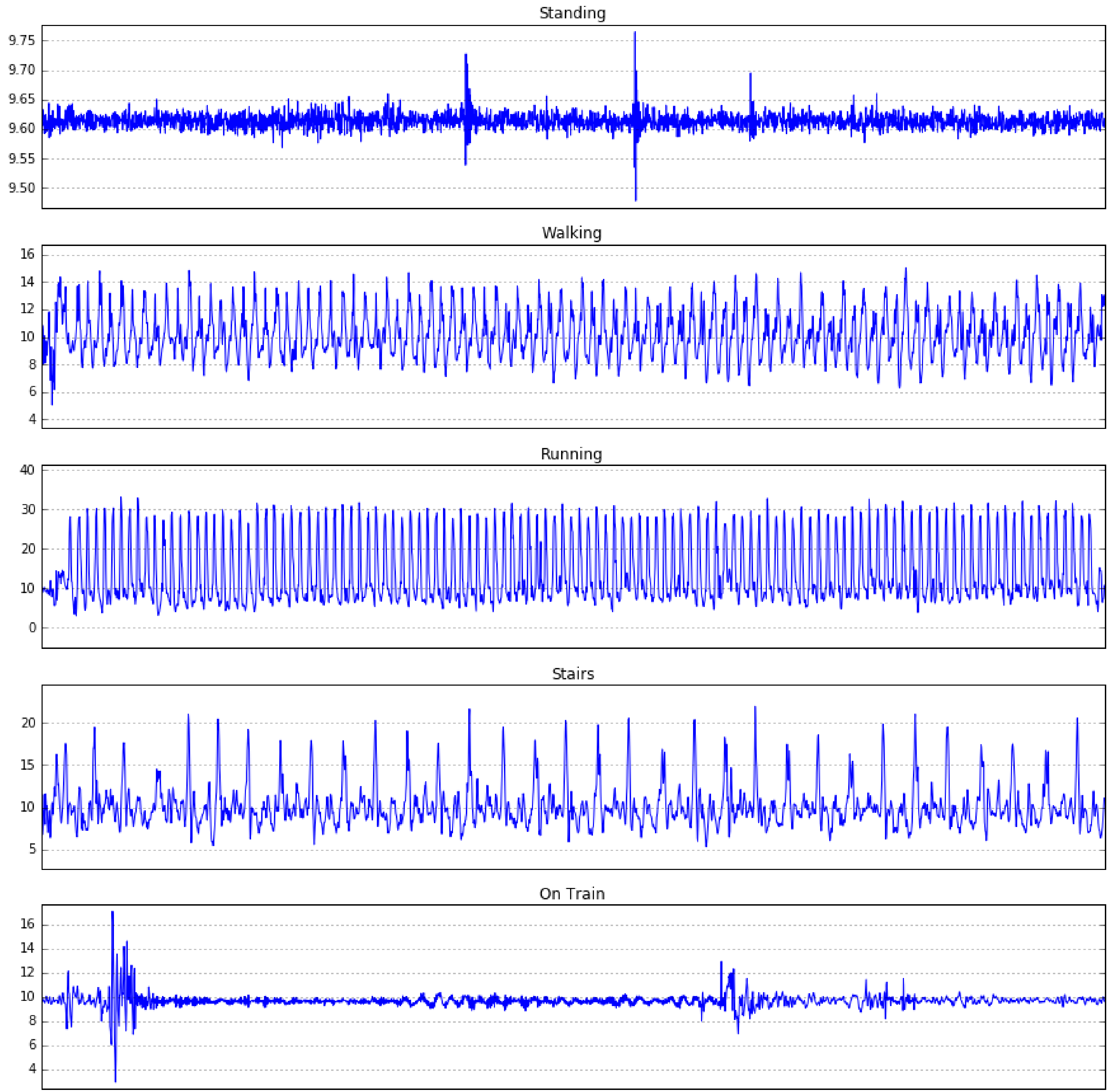


The magnitude vector

- We don't know how the phone is oriented
- We want to capture what is happening in the 3 axes in a single time series

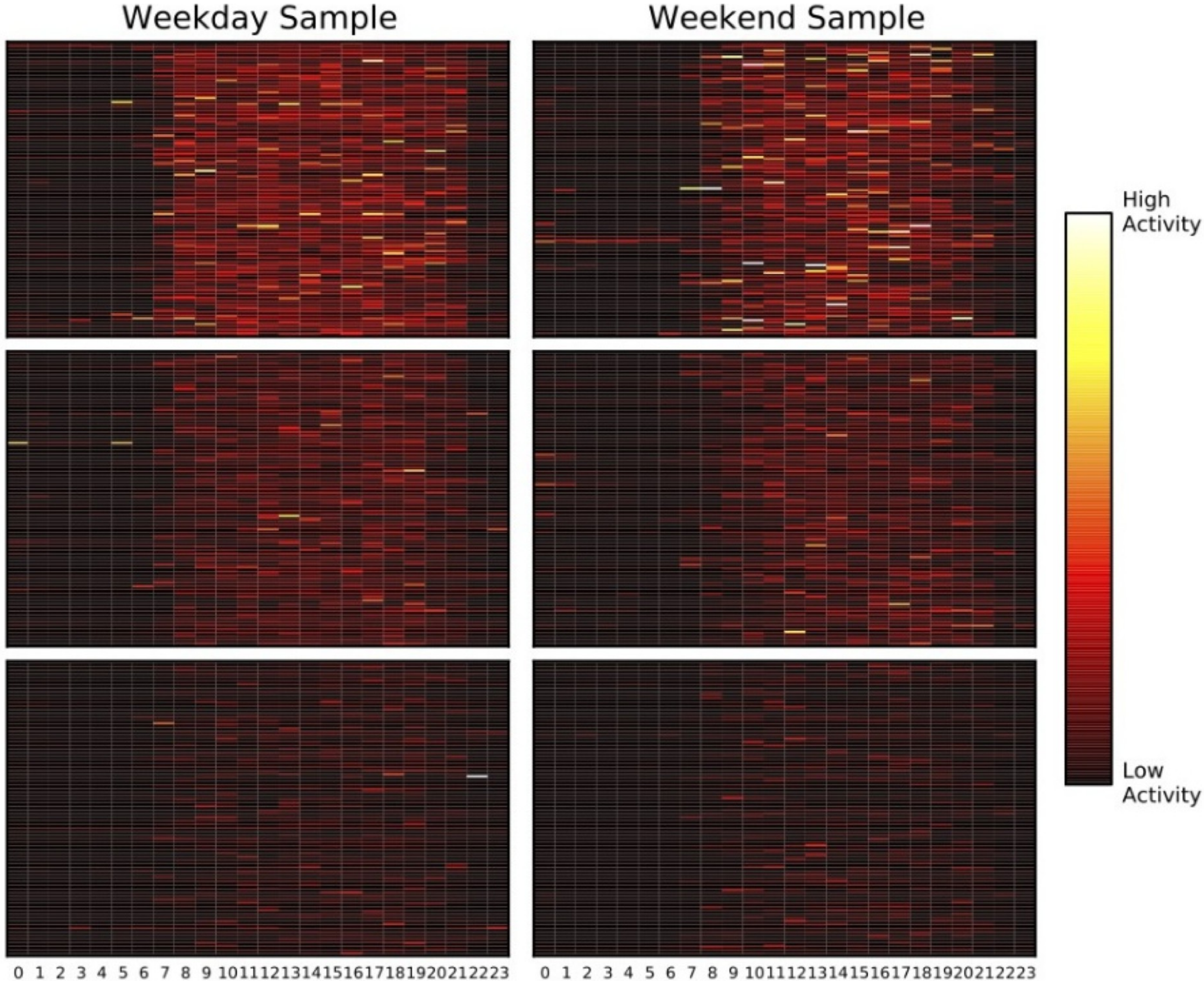
```
import math

def magnitude(activity):
    x2 = activity['xAxis'] * activity['xAxis']
    y2 = activity['yAxis'] * activity['yAxis']
    z2 = activity['zAxis'] * activity['zAxis']
    m2 = x2 + y2 + z2
    m = m2.apply(lambda x: math.sqrt(x))
    return m
```

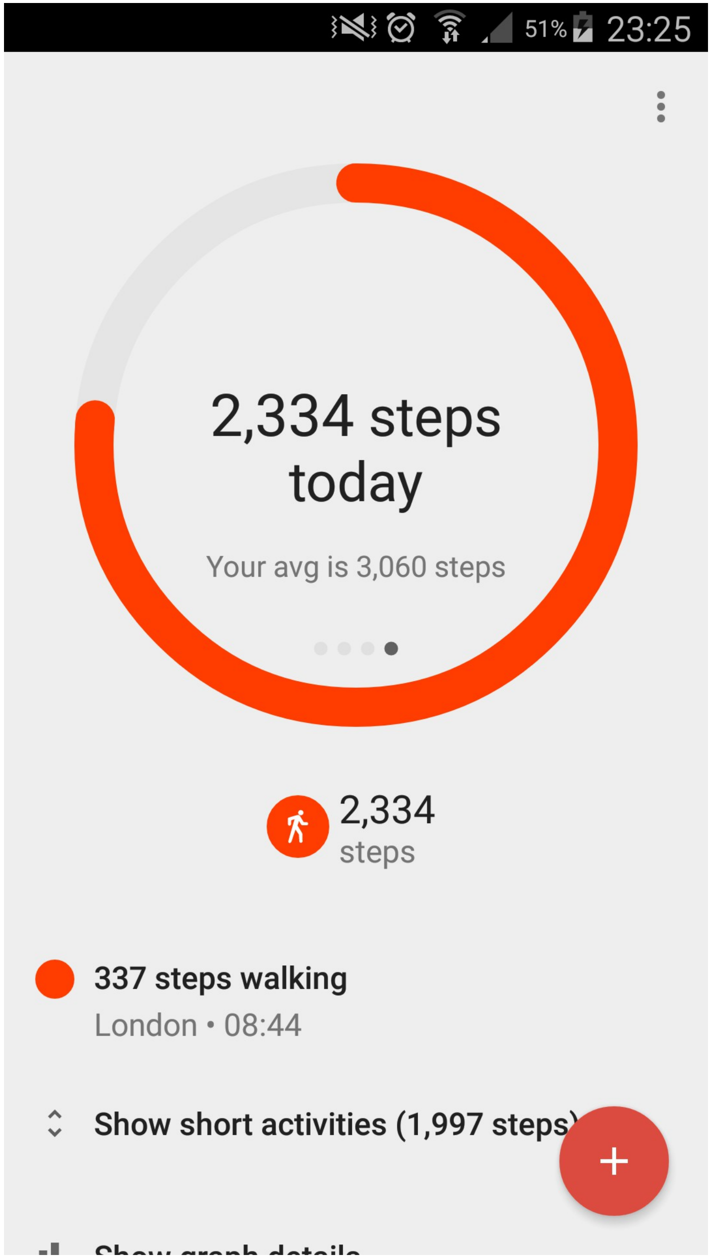

Standing
9.75
9.70
9.65
9.60
9.55
9.50
Walking
16
14
12
10
8
6
4
Running
40
30
20
10
0
Stairs
20
15
10
5
On Train
16
14
12
10
8
6
4

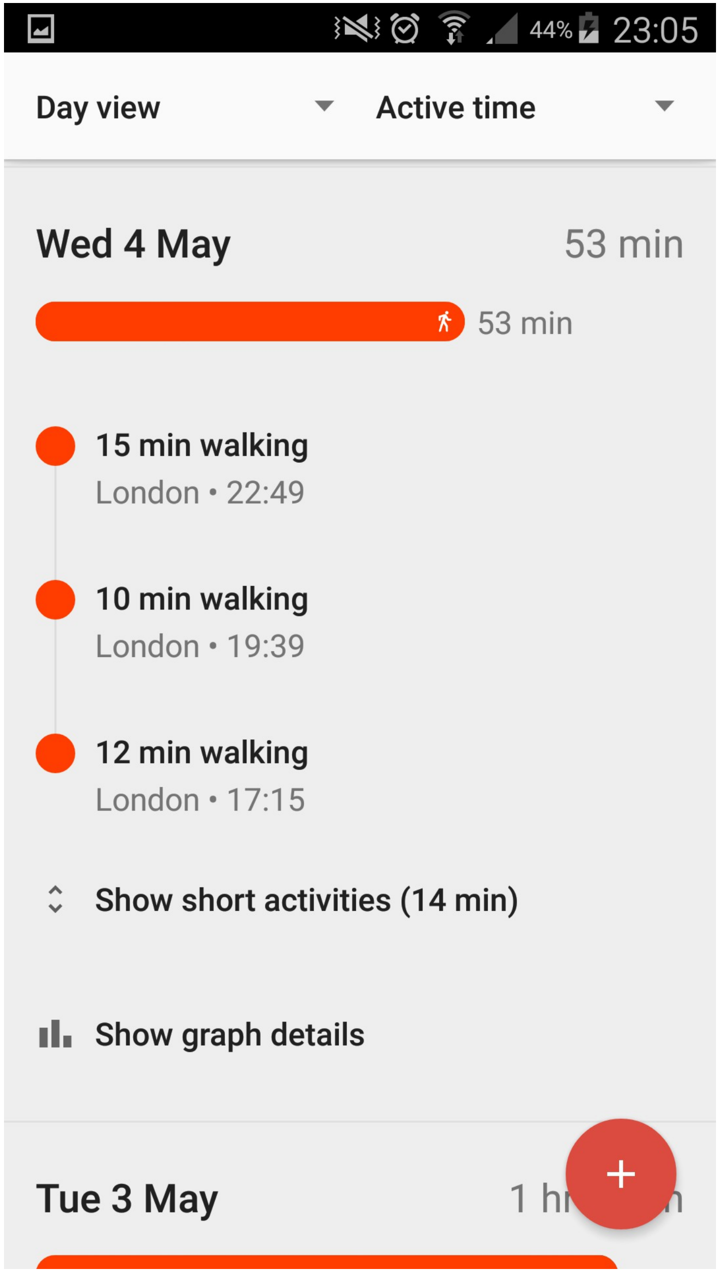
Applications

- Step counting
  - Brajdic, Harle. “Walk detection and step counting on unconstrainted smartphones.” ACM Ubicomp '13.
- Unsupervised learning (profiling)
  - Lathia et al. “Happy People Live Active Lives.” Under Submission.
- Activity classification


Weekday Sample
Weekend Sample
High Activity
Low Activity
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

Activity Classification

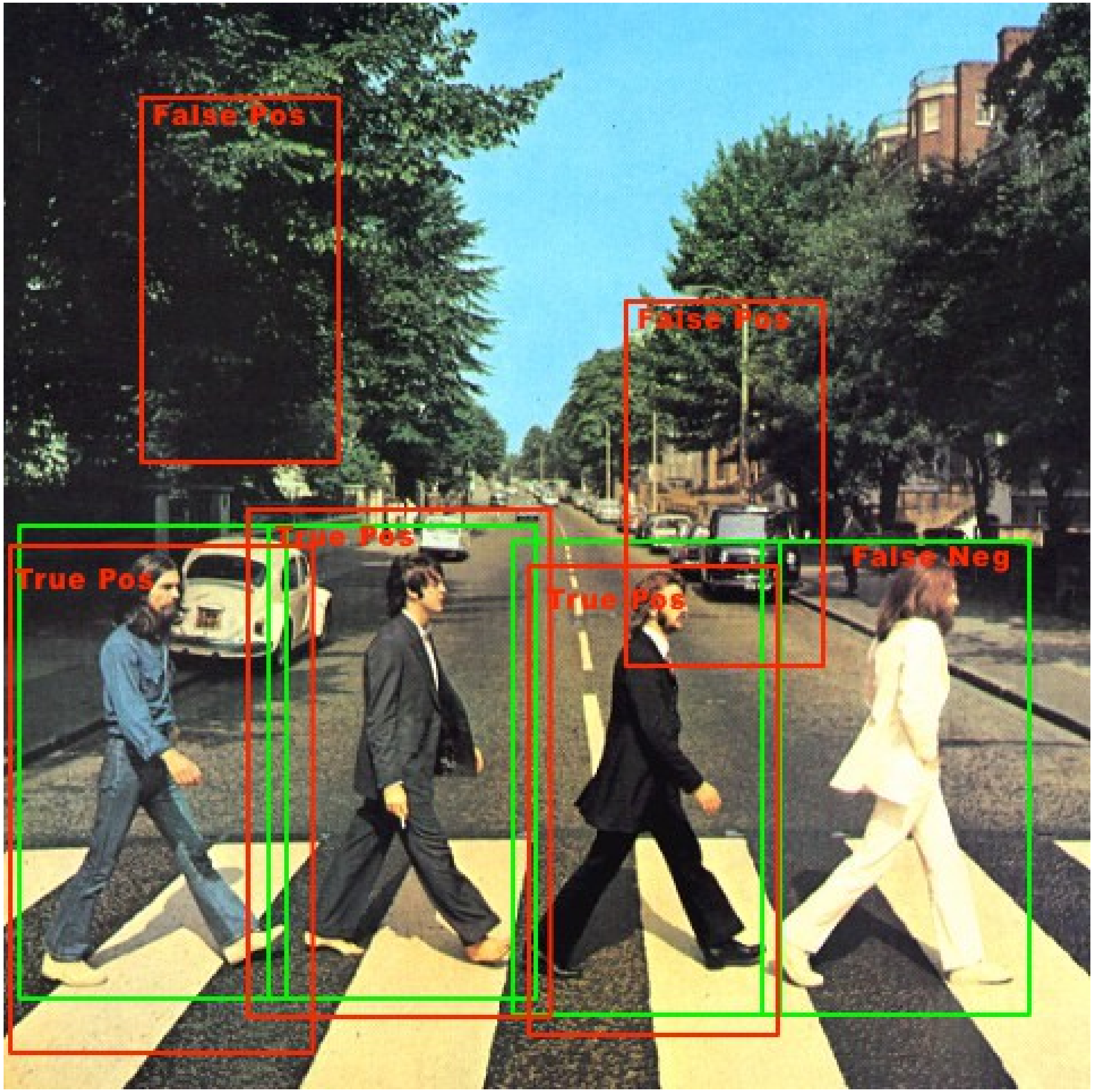




Activity classification: overview

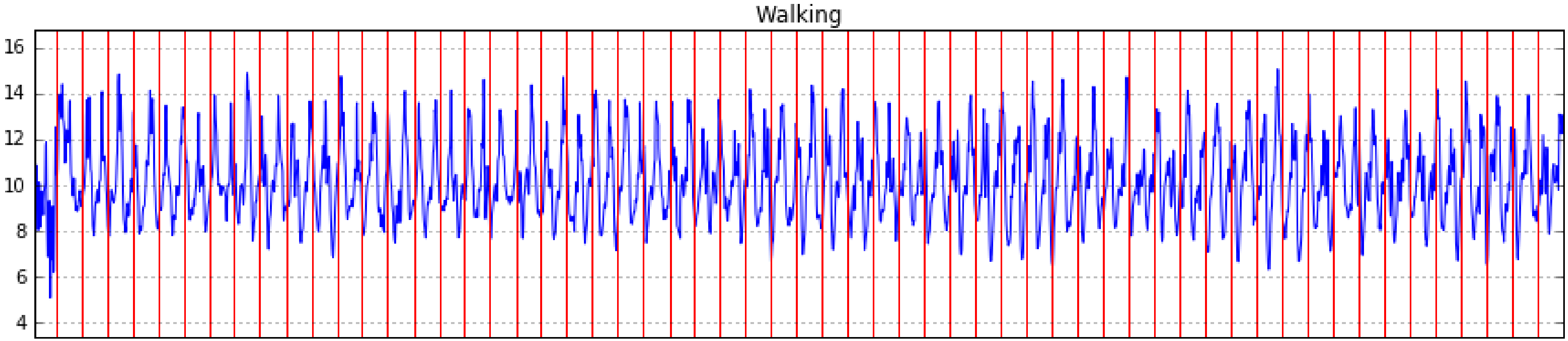
- Get the time series data into some way to train a classifier
- Train a classifier
- Predict activities
- ??
- Profit

Related Problem



Windowing

```
def windows(df, size=100):
    start = 0
    while start < df.count():
        yield start, start + size
        start += (size / 2)
```



- Extract features from each window

Extract features from windows

- Statistical (mean, std dev)
- Time-series (jitter, kurtosis)
- Signal (frequency)

Reading: Hemminki, Nurmi, Tarkoma. "Accelerometer-based Transportation Mode Detection on Smartphones." ACM Sensys '13.

Extract features from windows

```
def jitter(axis, start, end):
    j = float(0)
    for i in xrange(start, min(end, axis.count())):
        if start != 0:
            j += abs(axis[i] - axis[i-1])
    return j / (end-start)

def mean_crossing_rate(axis, start, end):
    cr = 0
    m = axis.mean()
    for i in xrange(start, min(end, axis.count())):
        if start != 0:
            p = axis[i-1] > m
            c = axis[i] > m
            if p != c:
                cr += 1
    return float(cr) / (end-start-1)
```

Extract features from windows

```
def window_summary(axis, start, end):
    acf = stattools.acf(axis[start:end])
    acv = stattools.acovf(axis[start:end])
    sqd_error = (axis[start:end] - axis[start:end].mean()) ** 2
    return [
        jitter(axis, start, end),
        mean_crossing_rate(axis, start, end),
        axis[start:end].mean(),
        axis[start:end].std(),
        axis[start:end].var(),
        axis[start:end].min(),
        axis[start:end].max(),
        acf.mean(), # mean auto correlation
        acf.std(), # standard deviation auto correlation
        acv.mean(), # mean auto covariance
        acv.std(), # standard deviation auto covariance
        skew(axis[start:end]),
        kurtosis(axis[start:end]),
        math.sqrt(sqd_error.mean())
    ]
```

```
from scipy.stats import skew, kurtosis
from statsmodels.tsa import stattools
```

Extract features from windows

```
def features(activity):
    for (start, end) in windows(activity['timestamp']):
        features = []
        for axis in ['xAxis', 'yAxis', 'zAxis', 'magnitude']:
            features += window_summary(activity[axis], start, end)
        yield features
```

Extract features from windows

```
def features(activity):
    for (start, end) in windows(activity['timestamp']):
        features = []
        for axis in ['xAxis', 'yAxis', 'zAxis', 'magnitude']:
```

2	0	0.0110193128	0.4444444444444444	0.24805126300000002	0.010428860696702456	0.00010876113543122523	0.21787234999999999
3	0	0.009211690299999951	0.37373737373737376	0.24554333720000002	0.009997528865246477	9.9950583411436501e-05	0.21787234999999999
4	0	0.008367732499999973	0.35353535353535354	0.24606407550000001	0.009385774134665736	8.809275610696035e-05	0.22146365000000001
5	0	0.010606313999999999	0.37373737373737376	0.24699781529999998	0.012171632439769342	0.00014814863624884538	0.21547815000000001
6	0	0.011665747799999997	0.43434343434343436	0.2469020502	0.012901450455182834	0.00016644742384753738	0.21547815000000001
7	0	0.010636240199999996	0.42424242424242425	0.2485001788	0.010698085897715799	0.00011444904187490564	0.22565351
8	0	0.010504559099999995	0.3333333333333333	0.2460999895	0.01346882131325247	0.00018140914756832399	0.20649988999999999

Label

Features

Data is ready.. classify

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.dummy import DummyClassifier
from sklearn.cross_validation import train_test_split

c = RandomForestClassifier()
b = DummyClassifier() # generates predictions by respecting the training set's class distribution

results = []
baselines = []

for i in range(0, 10):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)
    c.fit(X_train, y_train)
    b.fit(X_train, y_train)
    res = c.score(X_test, y_test)
    bas = b.score(X_test, y_test)
    print 'Loop', i, res, bas
    results.append(res)
    baselines.append(bas)

print '\nBaseline', np.mean(baselines), np.std(baselines)
print 'Random Forest', np.mean(results), np.std(results)
```

Data is ready.. classify

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.dummy import DummyClassifier
from sklearn.cross_validation import train_test_split

c = RandomForestClassifier()
b = DummyClassifier() # generates predictions by respecting the training set's class distribution

results = []
baselines = [

for i in rang
    X_train, X
    c.fit(X_t
    b.fit(X_t
    res = c.s
    bas = b.s
    print 'Lo
    results.a
    baselines

print '\nBase
print 'Random
```

```
Loop 0 0.966666666667 0.241666666667
Loop 1 0.991666666667 0.241666666667
Loop 2 0.975 0.191666666667
Loop 3 0.975 0.166666666667
Loop 4 0.983333333333 0.216666666667
Loop 5 0.975 0.208333333333
Loop 6 0.991666666667 0.25
Loop 7 0.975 0.208333333333
Loop 8 1.0 0.216666666667
Loop 9 0.975 0.2

Baseline 0.214166666667 0.0241666666667
Random Forest 0.980833333333 0.00989528507253
```

Further thoughts

- Collecting data efficiently
  - Background processes use loads of battery
- Real data is messier
  - This was one person, one phone
- Feature engineering
  - This was just an example.
- Other flavours of classification
  - Binary: “Is this walking?”
  - Personalized vs. global models

Conclusion

- Collecting accelerometer data
- A peek at the raw data
- Magnitude data
- Applications
- Feature extraction
- Focus on classification

Mining Smartphone Data (with Python)

@neal_lathia

PyData London 2016

https://github.com/nlathia/pydata_2016