

Mining Smartphone Data (with Python)

@neal_lathia
PyData London 2016



Smartphones have sensors!

- Accelerometer (acceleration)
 - Gyroscope (orientation)
 - GPS, Wi-Fi (location)
 - ...
-
- Microphone (sound)
 - Bluetooth (co-location)

Smartphones have sensors!

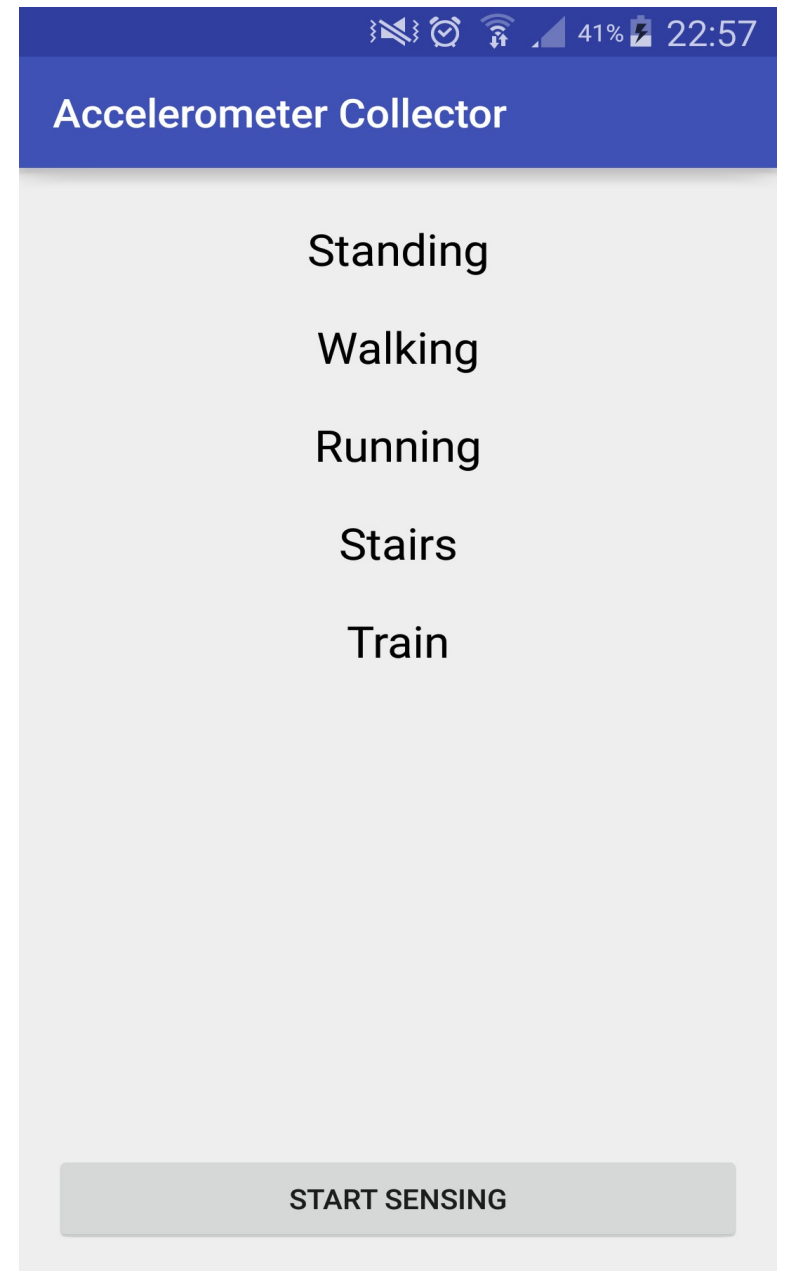
- Accelerometer (acceleration)
- Gyroscope (orientation)
- GPS, Wi-Fi (location)

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

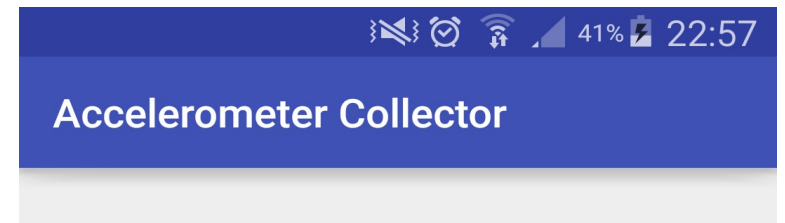
This talk

- 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

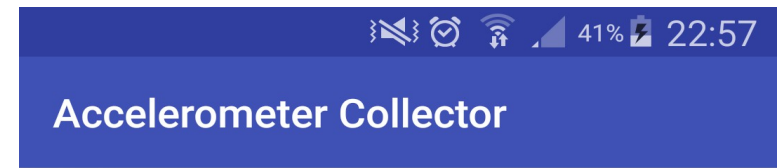


```
public void start(final Context context) throws IOException
{
    fileWriter = new DataWriter(context, LabelPreferences.getLabel(context));
    mSensorManager.registerListener(this, mSensor, SensorManager.SENSOR_DELAY_GAME);
    isSensing = true;
}

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

START SENSING

Collecting Data



A screenshot of an Android file manager application. The top status bar shows the time as 12:00 and the device name as GT-I9505. The file manager interface has a top bar with navigation buttons (back, forward) and a search bar. Below this, a list of files and folders is displayed. A folder named 'AccelerometerData' is expanded, showing a list of CSV files. The files are: Running_1462487326006.csv, Stairs_1462487242397.csv, Standing_1462486338643.csv, Standing_1462486748955.csv, Standing_1462486804782.csv, Standing_1462518268451.csv, Standing_1462532258375.csv, Train_1462518004872.csv, Train_1462518152855.csv, Train_1462518289511.csv, and Walking_1462487070722.csv. The dates next to the files range from 05/05/2016 to 06/05/2016.

```
    io(context));  
    SENSOR_DELAY_GAME);
```

START SENSING

The raw data

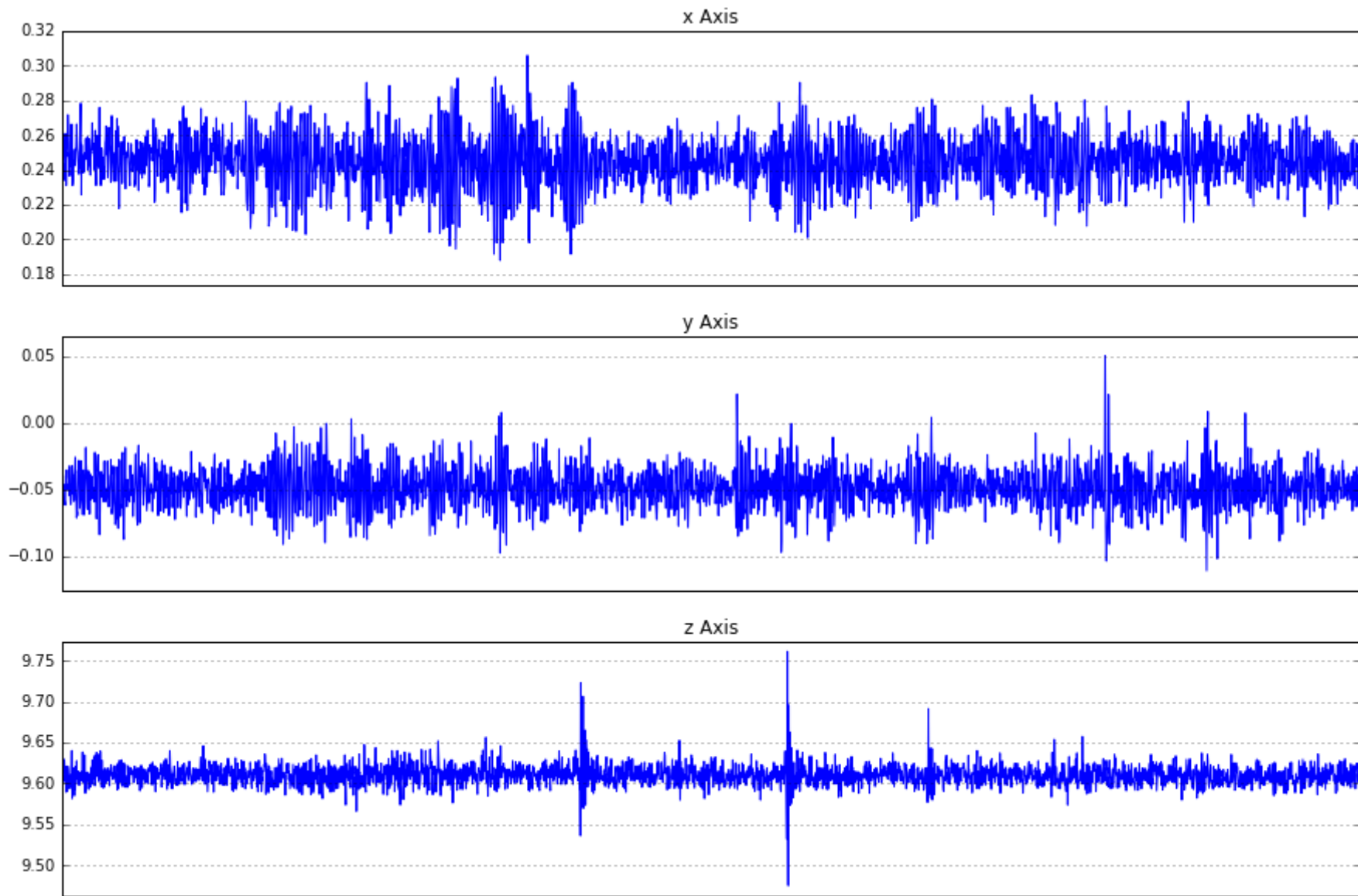
```
STANDING = pd.read_csv('../Data/Standing_1462486804782.csv', header=0)
WALKING = pd.read_csv('../Data/Walking_1462487070722.csv', header=0)
RUNNING = pd.read_csv('../Data/Running_1462487326006.csv', header=0)
STAIRS = pd.read_csv('../Data/Stairs_1462487242397.csv', header=0)
ON_TRAIN = pd.read_csv('../Data/Train_1462518004872.csv', header=0)

STANDING.head()
```

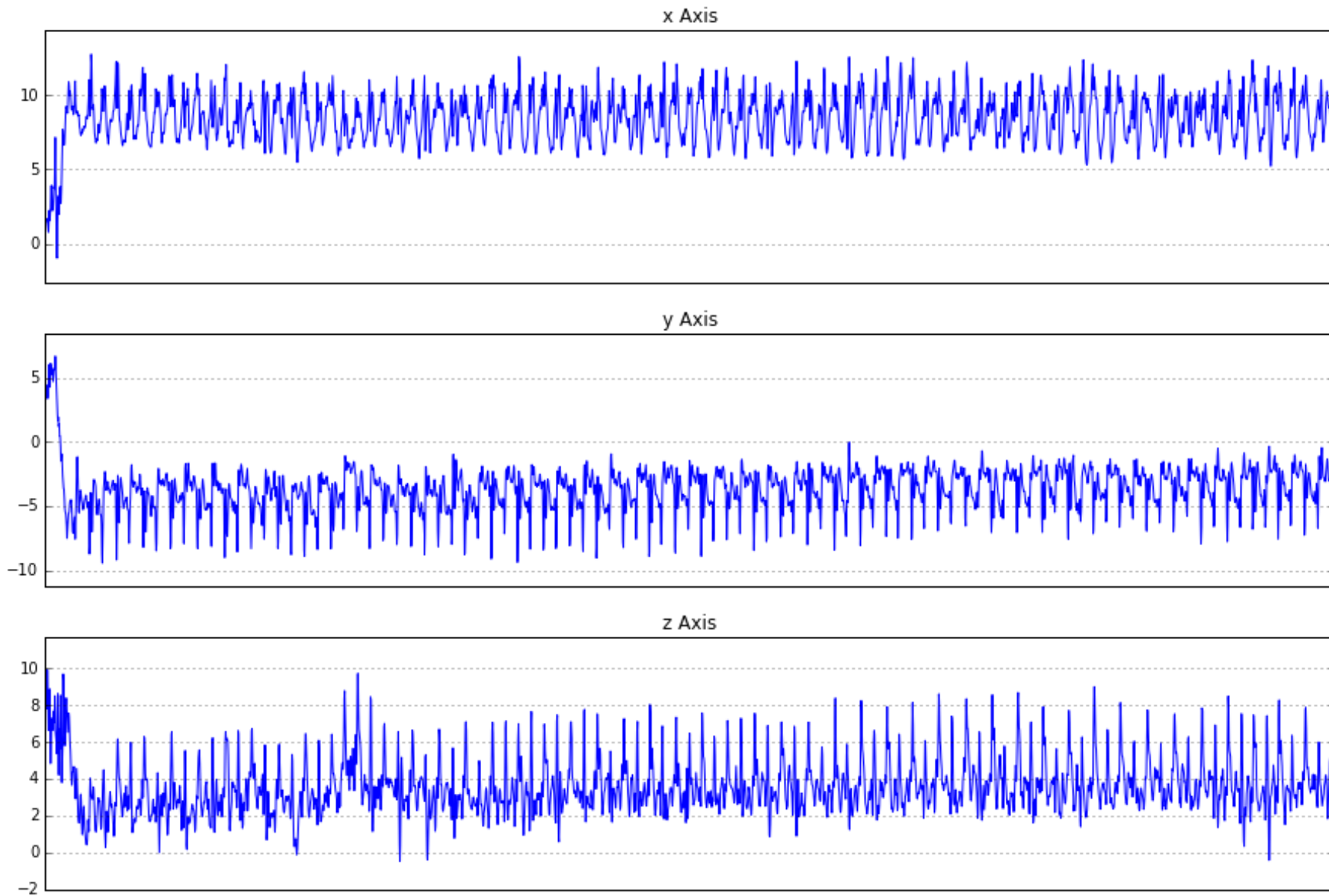
]:

	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

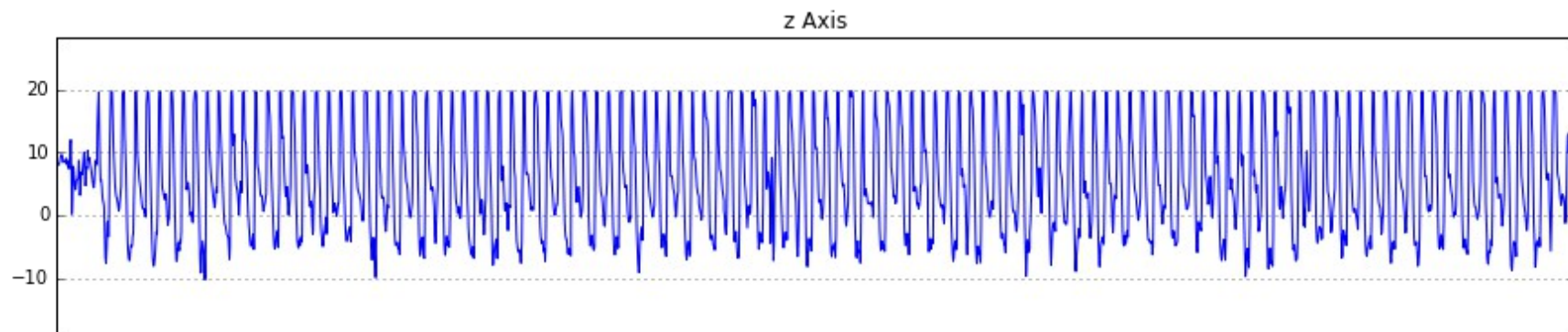
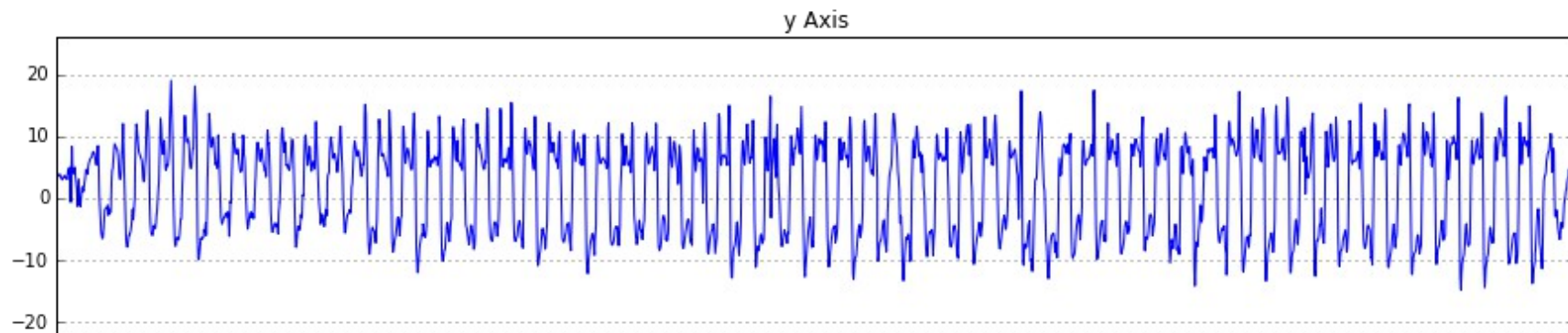
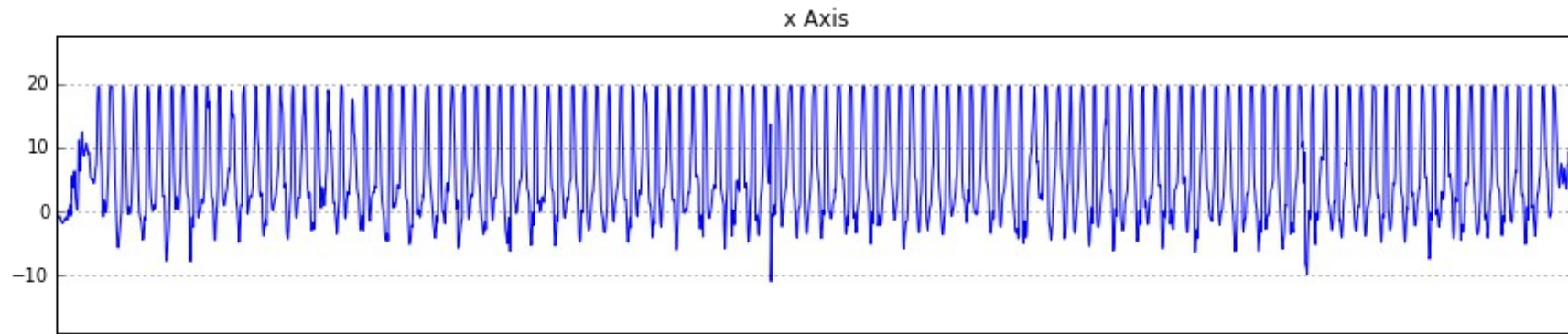
Raw: Walk



Raw: Walk

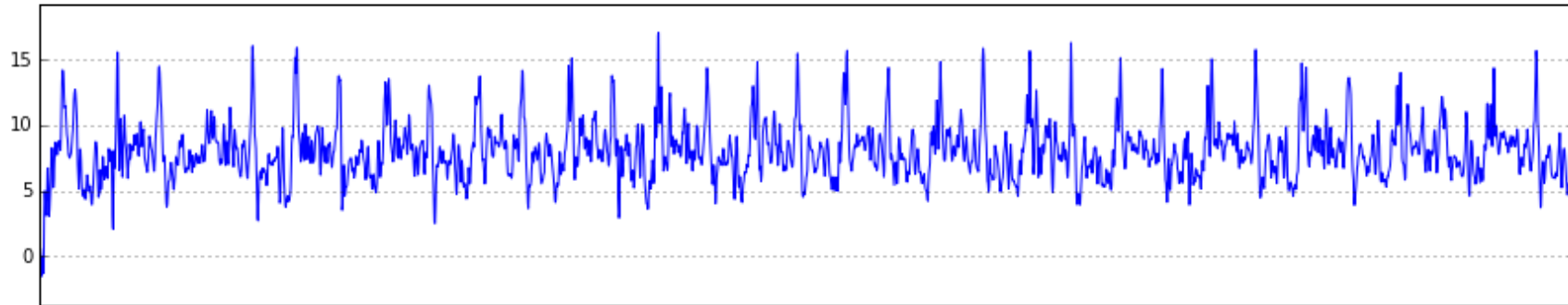


Raw: Run

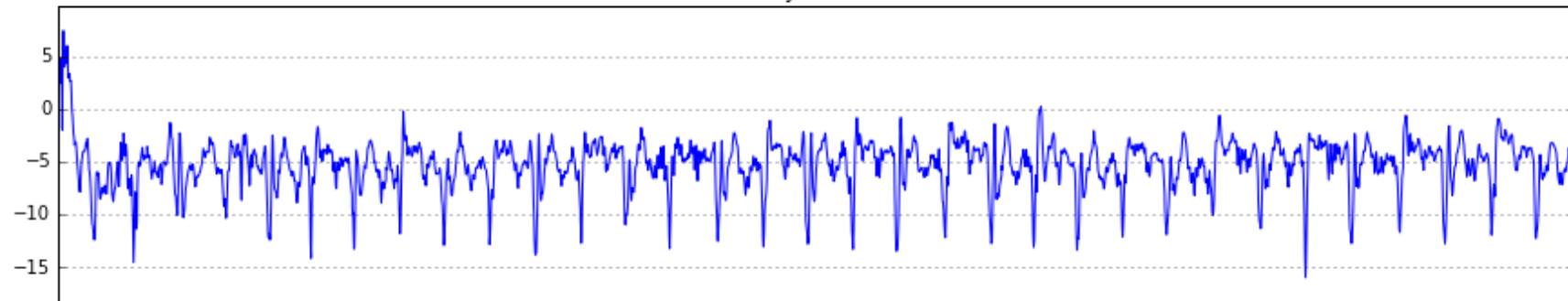


Raw: Stairs

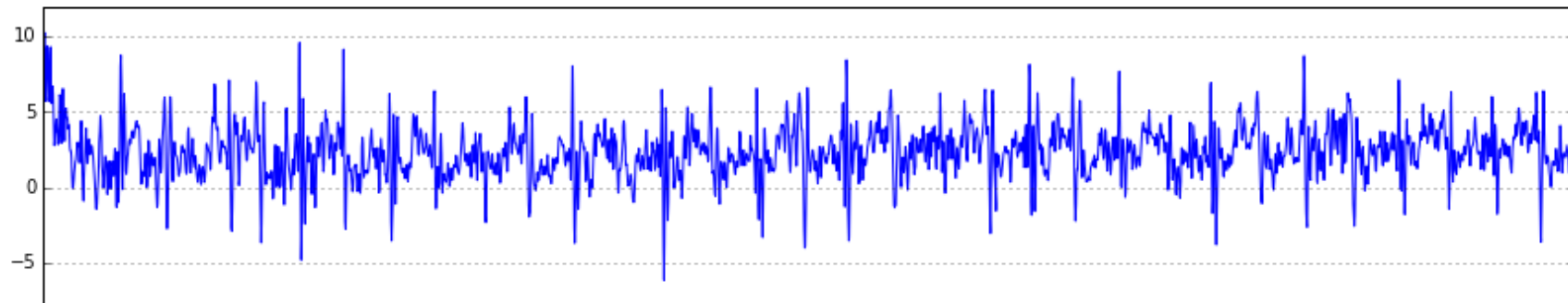
x Axis



y Axis

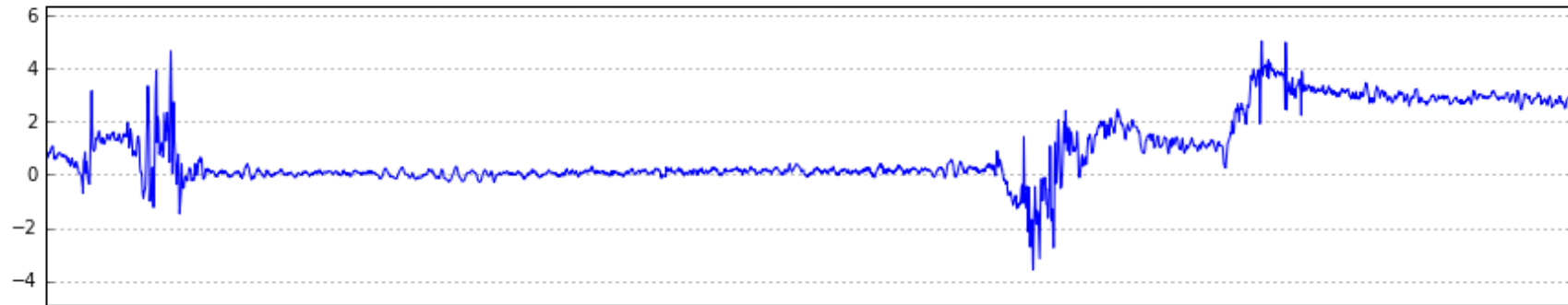


z Axis

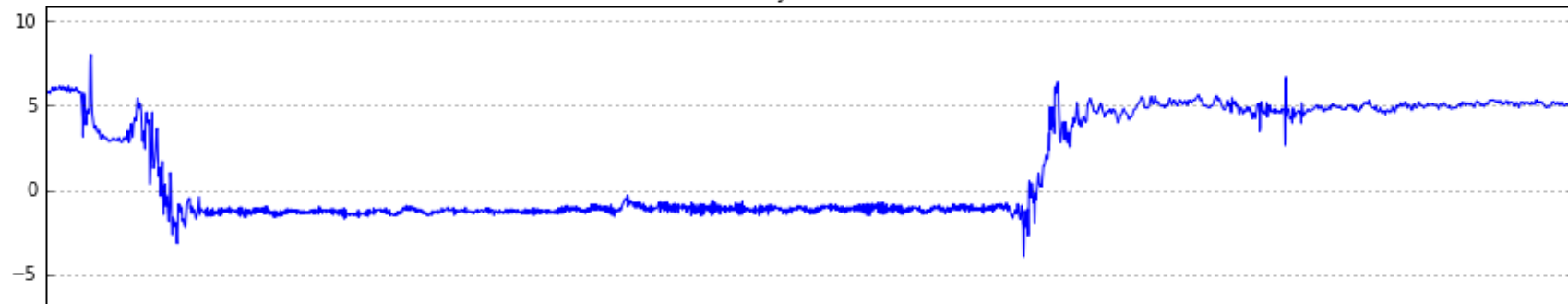


Raw: Stairs

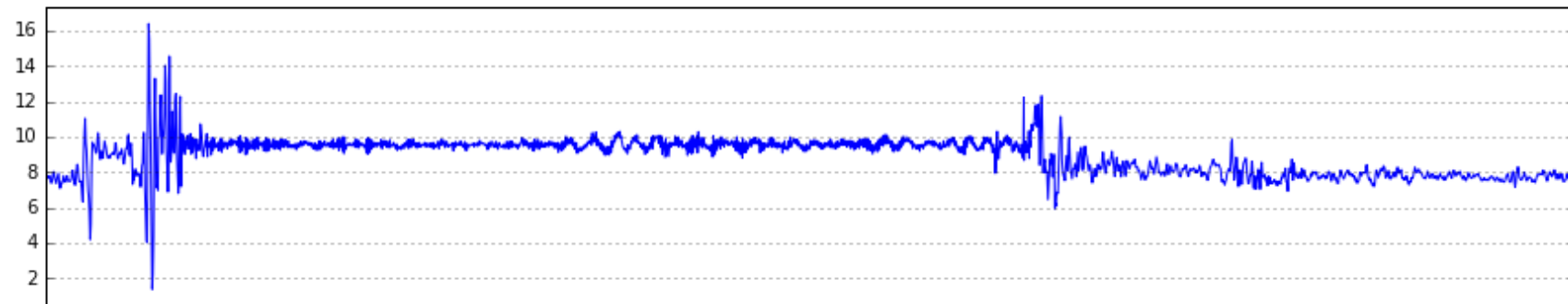
x Axis



y Axis



z Axis



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
```

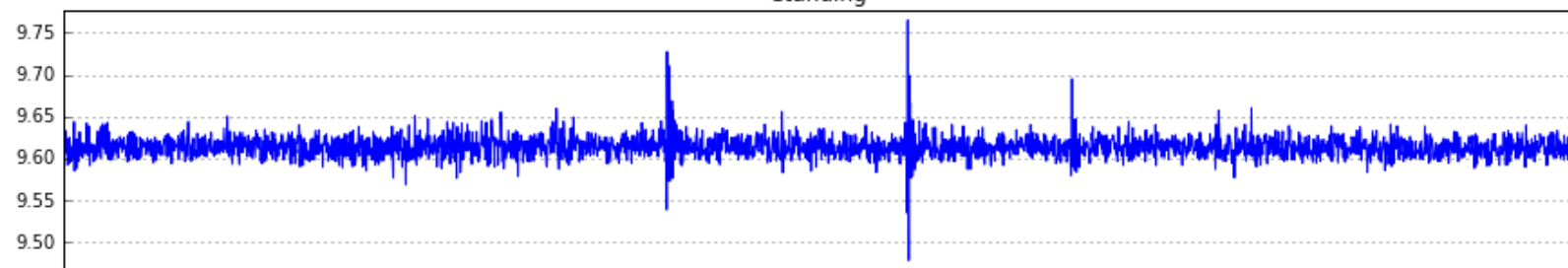

The magnitude vector

- 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

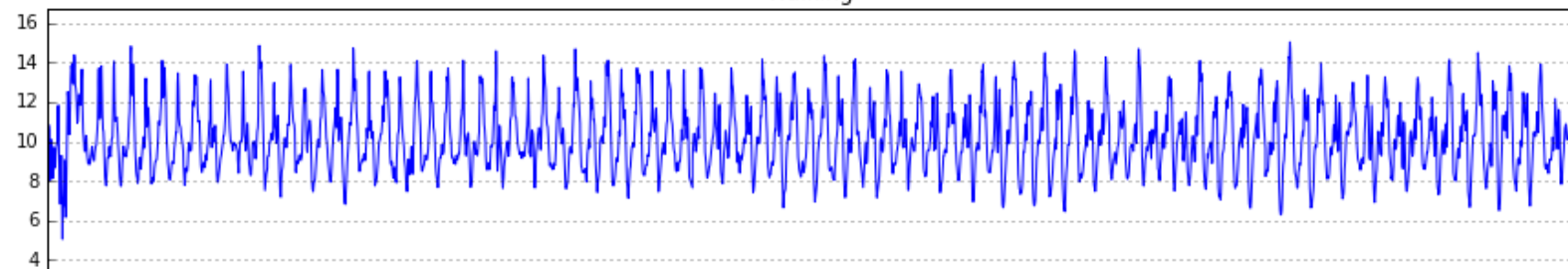
```
import math

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
```

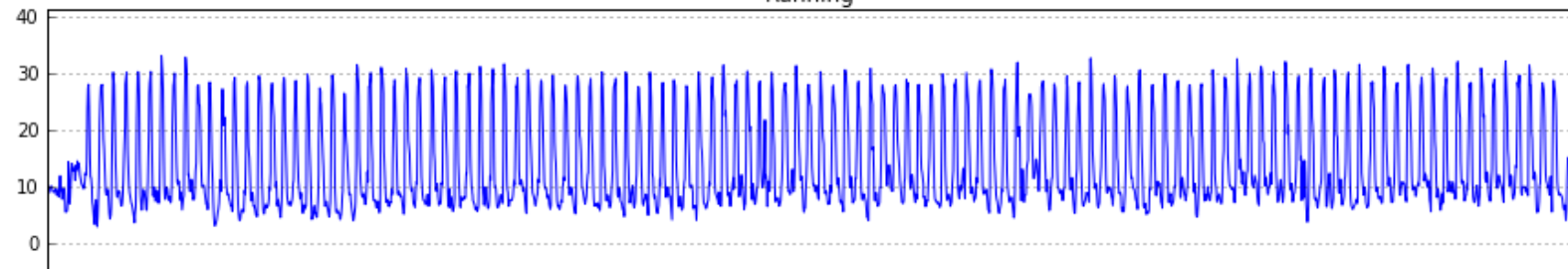
Standing



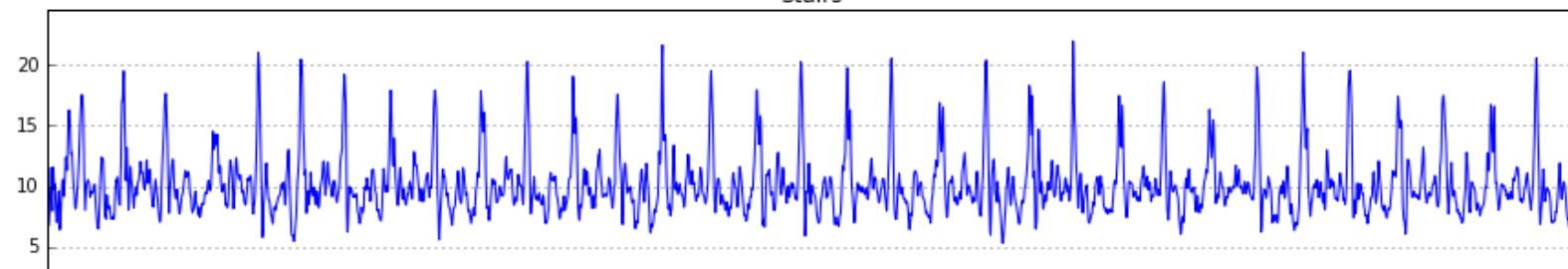
Walking



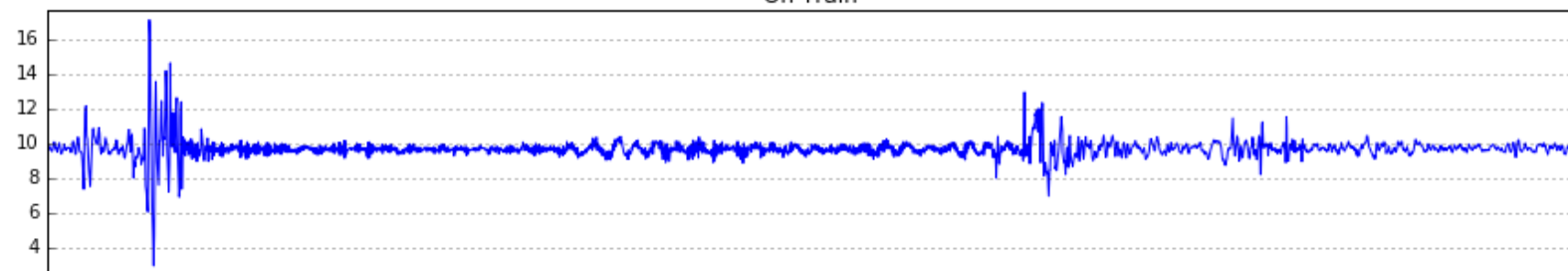
Running



Stairs



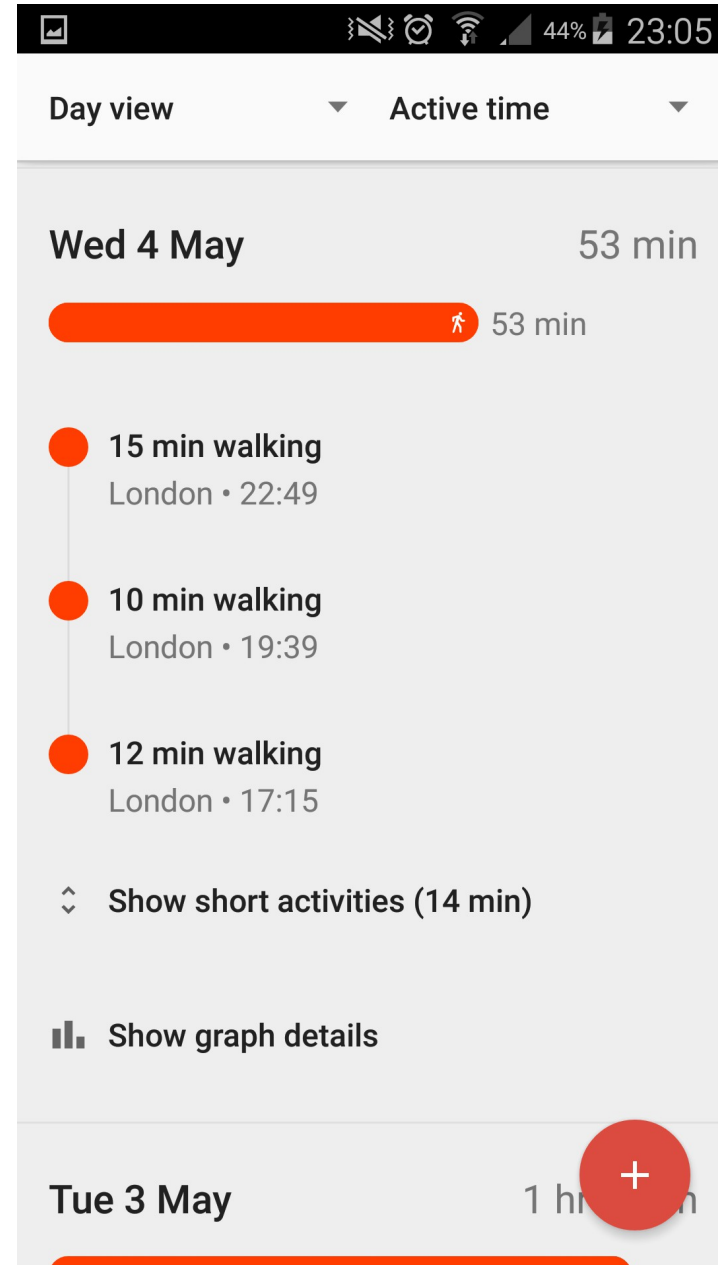
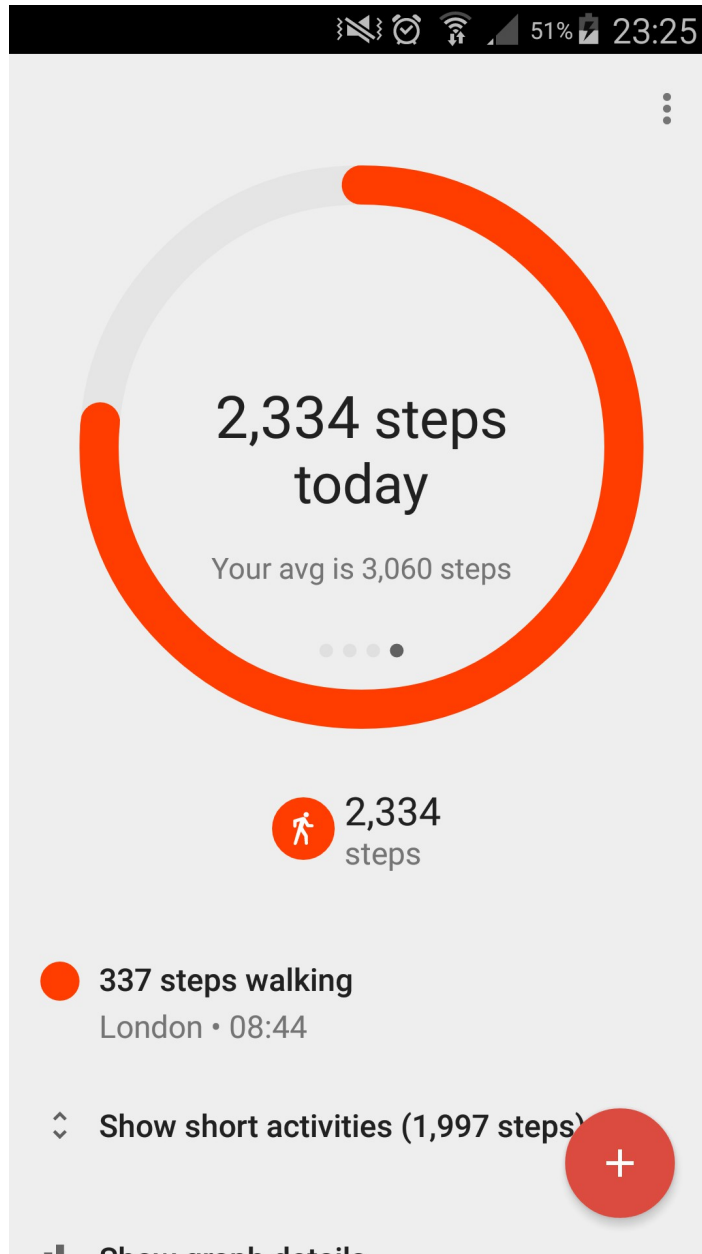
On Train



Applications

- Step counting
- Unsupervised learning (profiling)
- Activity classification

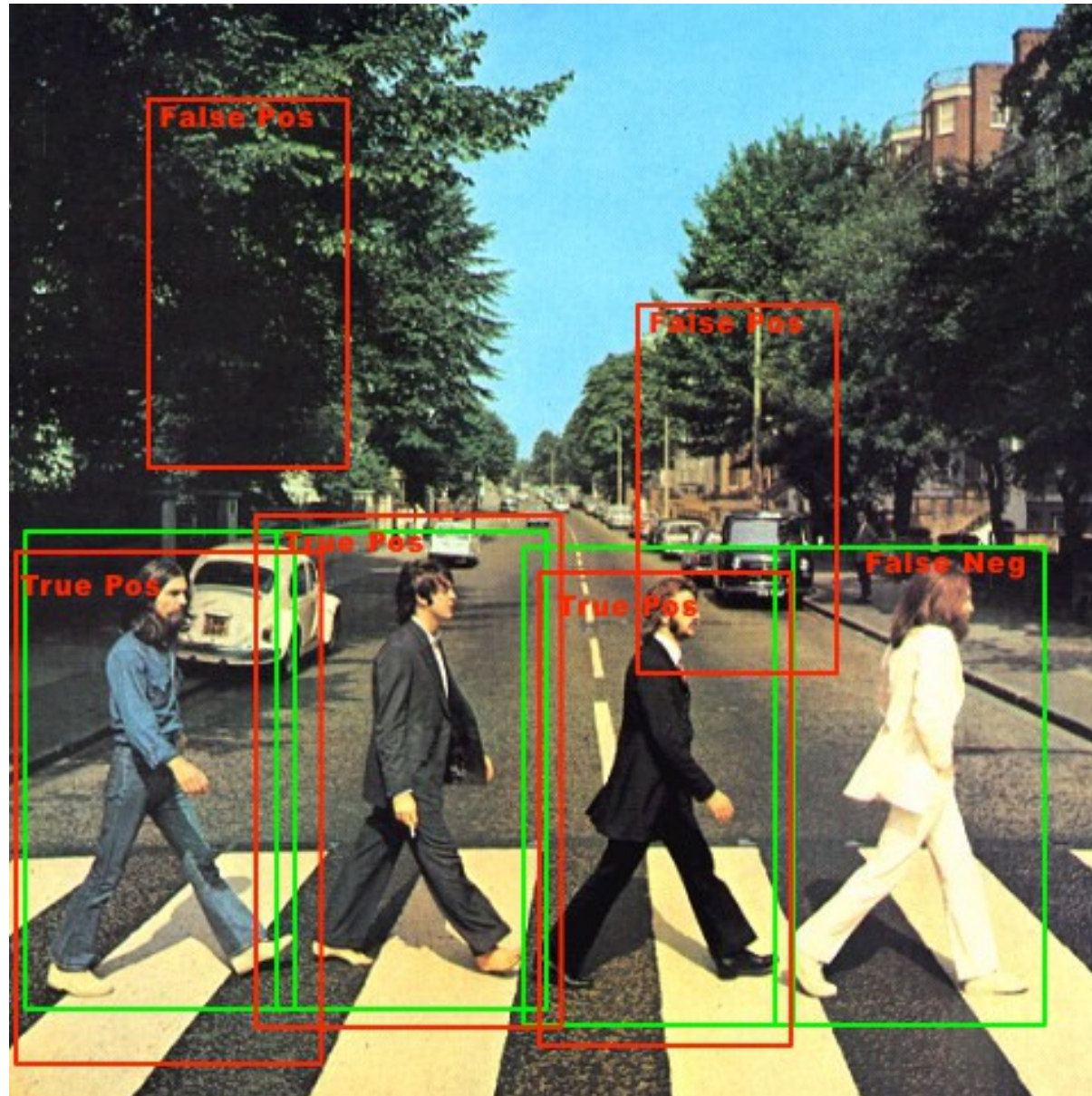
Applications



Activity classification: overview

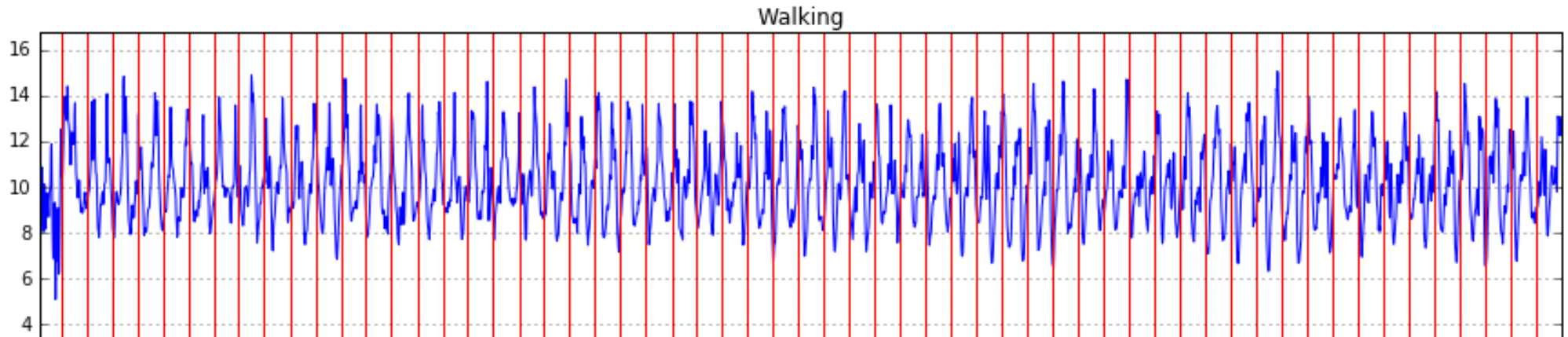
- Get the time series data into some way to train a classifier
- Train a classifier
- Predict activities
- ??
- Profit

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

Reading: Hemminki, Nurmi, Tarkoma.

“Accelerometer-based Transportation Mode Detection on Smartphones.” ACM Sensys '13.

Extract features from windows

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

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())  
    ]
```

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
```

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 range(10):
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=i)
    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 %d %f %f' % (i, res, bas)
    results.append((i, res, bas))
    baselines.append(bas)

print '\nBaseline %f %f' % (baselines[0], baselines[-1])
print 'Random Forest %f %f' % (results[-1][1], results[-1][2])
```

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	0.024166666667
Random Forest	0.980833333333	0.00989528507253

Further thoughts

- Collecting data efficiently
 - Background processes use loads of battery
- Feature engineering
 - This was just an example.
- Other flavours of classification
 - E.g. binary: “Is this walking?”

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