Smartphone sensing

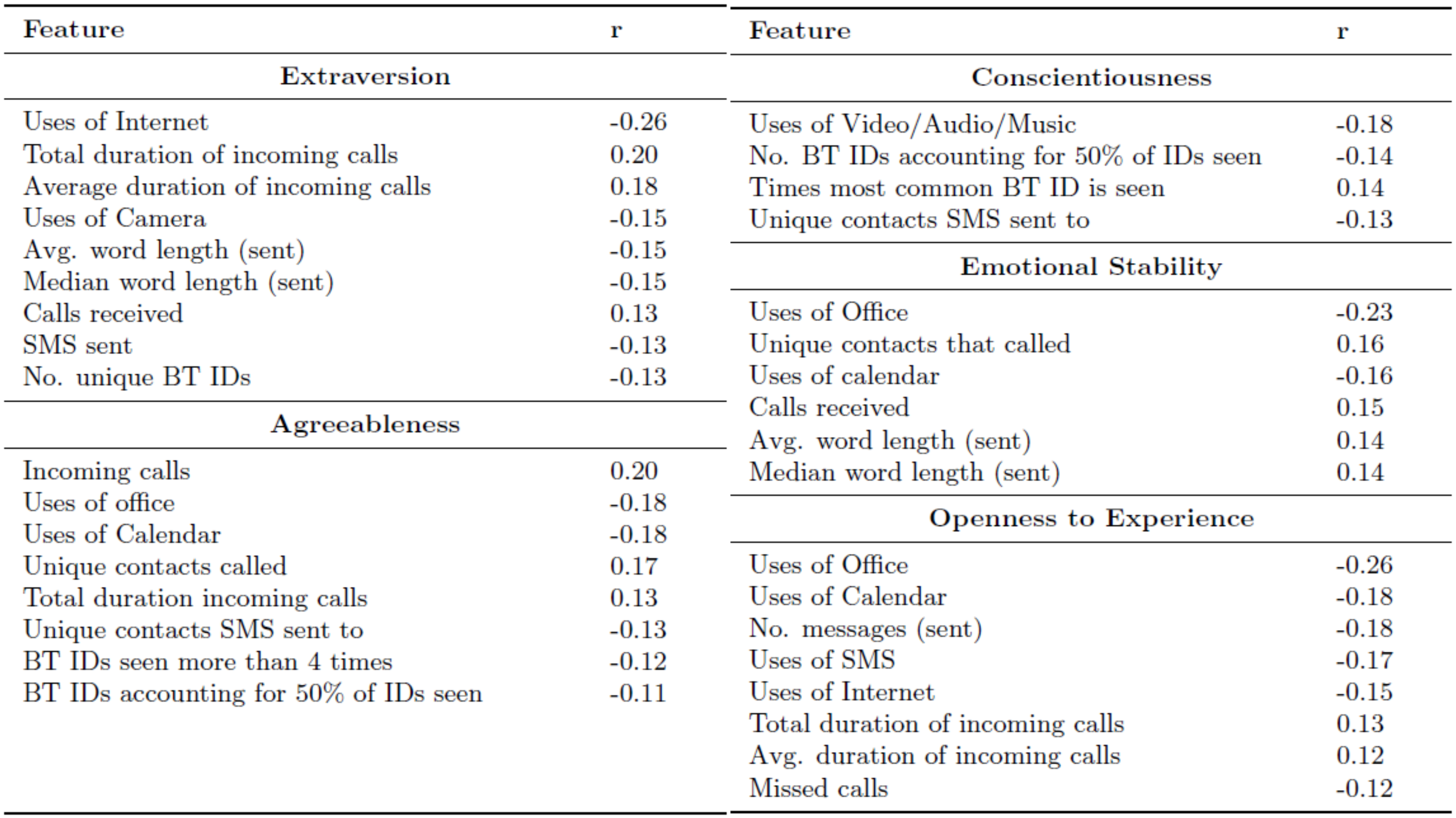
10 November 2014

Urban Noise Pollution

- Example project: NoiseTube
  http://noisetube.net
- Started at the Sony Computer Science Lab in Paris and currently hosted by the Vrije Universiteit Brussel.
- Mobile app turns smartphones into noise sensors:
  - measure sound exposure in everyday environments
  - geolocalized measurement data
- Software released under the GNU LGPL v2.1 open source license
- Researcher access to (anonymized) collective noise data

Classifying Personality Traits

Who's Who with Big-Five: Analyzing and Classifying Personality Traits with Smartphones

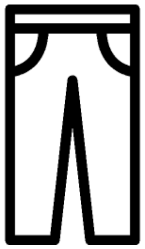
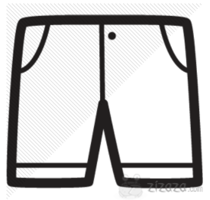


Feature	r
Extraversion	
Uses of Internet	-0.26
Total duration of incoming calls	0.20
Average duration of incoming calls	0.18
Uses of Camera	-0.15
Avg. word length (sent)	-0.15
Median word length (sent)	-0.15
Calls received	0.13
SMS sent	-0.13
No. unique BT IDs	-0.13
Agreeableness	
Incoming calls	0.20
Uses of office	-0.18
Uses of Calendar	-0.18
Unique contacts called	0.17
Total duration incoming calls	0.13
Unique contacts SMS sent to	-0.13
BT IDs seen more than 4 times	-0.12
BT IDs accounting for 50% of IDs seen	-0.11

Feature	r
Conscientiousness	
Uses of Video/Audio/Music	-0.18
No. BT IDs accounting for 50% of IDs seen	-0.14
Times most common BT ID is seen	0.14
Unique contacts SMS sent to	-0.13
Emotional Stability	
Uses of Office	-0.23
Unique contacts that called	0.16
Uses of calendar	-0.16
Calls received	0.15
Avg. word length (sent)	0.14
Median word length (sent)	0.14
Openness to Experience	
Uses of Office	-0.26
Uses of Calendar	-0.18
No. messages (sent)	-0.18
Uses of SMS	-0.17
Uses of Internet	-0.15
Total duration of incoming calls	0.13
Avg. duration of incoming calls	0.12
Missed calls	-0.12

What are you wearing today?

Huy Tran and Thanh Dang.
Clothing classification with smart phones.
In Proceedings of the 2014 ACM International Symposium on Wearable Computers: Adjunct Program (ISWC '14 Adjunct).
http://dang.encs.vancouver.wsu.edu/pubs/papers/ubicomp14.pdf

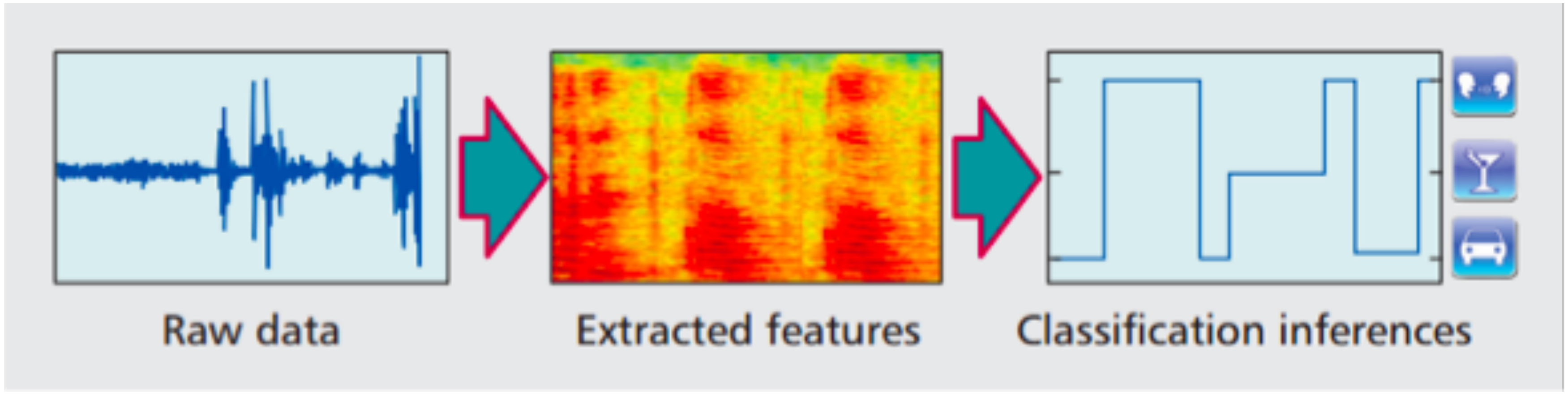
- Classification based on thermal insulation
- Use ambient data from smartphone in user pocket: relative humidity + temperature
- Ambient data sample at 2Hz for 5 minutes
- 70% accuracy

Sensor data analysis

- Server-based:
  - Necessary for resource-intensive tasks
  - Data transfer: energy/monetary cost, latency, security, privacy
- Device-based:
  - Requires fast and lightweight methods
  - Battery consumption
  - Privacy concerns

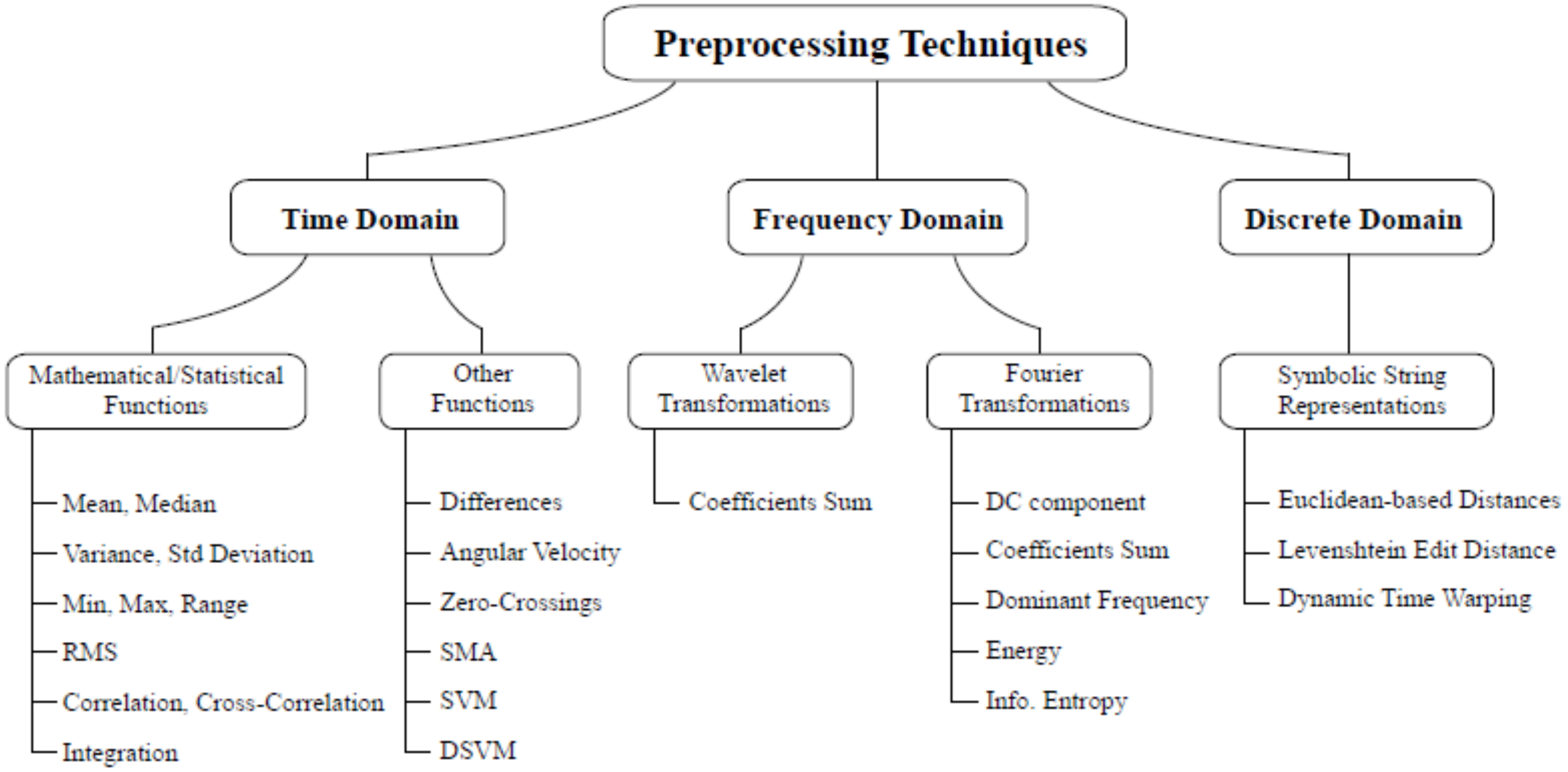
Sensor data analysis

- Common phases:
  1. Data acquisition
  2. Signal processing
  3. Feature extraction
  4. Classification



Features extraction

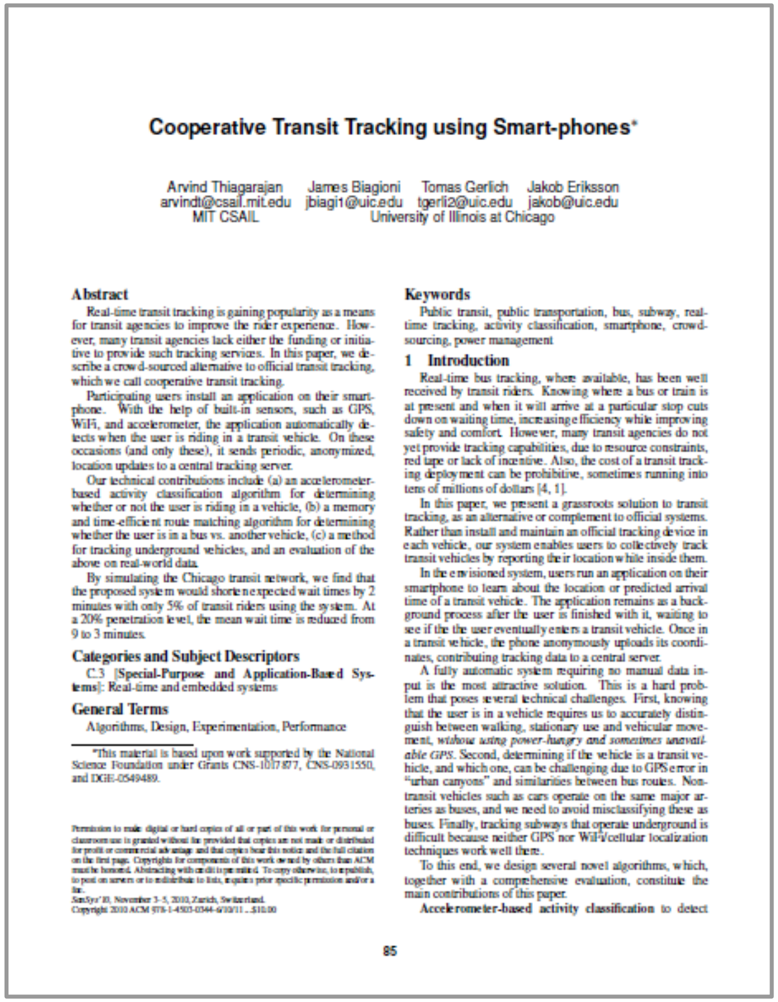
- Extract **information** from raw data



Classify Activity & Transportation Modes

- Accelerometer data can be used to classify a user activities:
  - Running, Walking, Stationary
  - Low power
- Combining motion classification with GPS tracking can recognize the user's mode of transportation:
  - Subway, bike, bus, car, walk…
  - GPS is power-hungry (400 mW)

An example from the literature



Cooperative Transit Tracking using Smart-phones*

Arvind Thiagarajan arvindt@csail.mit.edu MIT CSAIL
James Biagioni jbiagi1@uic.edu
Tomas Gerlich tgerli2@uic.edu
Jakob Eriksson jakob@uic.edu
University of Illinois at Chicago

Abstract

Real-time transit tracking is gaining popularity as a means for transit agencies to improve the rider experience. However, many transit agencies lack either the funding or initiative to provide such tracking services. In this paper, we describe a crowd-sourced alternative to official transit tracking, which we call cooperative transit tracking.

Participating users install an application on their smartphone. With the help of built-in sensors, such as GPS, WiFi, and accelerometer, the application automatically detects when the user is riding in a transit vehicle. On these occasions (and only these), it sends periodic, anonymized, location updates to a central tracking server.

Our technical contributions include (a) an accelerometer-based activity classification algorithm for determining whether or not the user is riding in a vehicle, (b) a memory and time-efficient route matching algorithm for determining whether the user is in a bus vs. another vehicle, (c) a method for tracking underground vehicles, and an evaluation of the above on real-world data.

By simulating the Chicago transit network, we find that the proposed system would shorten expected wait times by 2 minutes with only 5% of transit riders using the system. At a 20% penetration level, the mean wait time is reduced from 9 to 3 minutes.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Algorithms, Design, Experimentation, Performance

*This material is based upon work supported by the National Science Foundation under Grants CNS-1017877, CNS-0931550, and DGE-0549489.

Keywords

Public transit, public transportation, bus, subway, real-time tracking, activity classification, smartphone, crowd-sourcing, power management

1 Introduction

Real-time bus tracking, where available, has been well received by transit riders. Knowing where a bus or train is at present and when it will arrive at a particular stop cuts down on waiting time, increasing efficiency while improving safety and comfort. However, many transit agencies do not yet provide tracking capabilities, due to resource constraints, red tape or lack of incentive. Also, the cost of a transit tracking deployment can be prohibitive, sometimes running into tens of millions of dollars [4, 1].

In this paper, we present a grassroots solution to transit tracking, as an alternative or complement to official systems. Rather than install and maintain an official tracking device in each vehicle, our system enables users to collectively track transit vehicles by reporting their location while inside them.

In the envisioned system, users run an application on their smartphone to learn about the location or predicted arrival time of a transit vehicle. The application remains as a background process after the user is finished with it, waiting to see if the the user eventually enters a transit vehicle. Once in a transit vehicle, the phone anonymously uploads its coordinates, contributing tracking data to a central server.

A fully automatic system requiring no manual data input is the most attractive solution. This is a hard problem that poses several technical challenges. First, knowing that the user is in a vehicle requires us to accurately distinguish between walking, stationary use and vehicular movement, *without using power-hungry and sometimes unavailable GPS*. Second, determining if the vehicle is a transit vehicle, and which one, can be challenging due to GPS error in "urban canyons" and similarities between bus routes. Non-transit vehicles such as cars operate on the same major arteries as buses, and we need to avoid misclassifying these as buses. Finally, tracking subways that operate underground is difficult because neither GPS nor WiFi/cellular localization techniques work well there.

To this end, we design several novel algorithms, which, together with a comprehensive evaluation, constitute the main contributions of this paper.

Accelerometer-based activity classification to detect

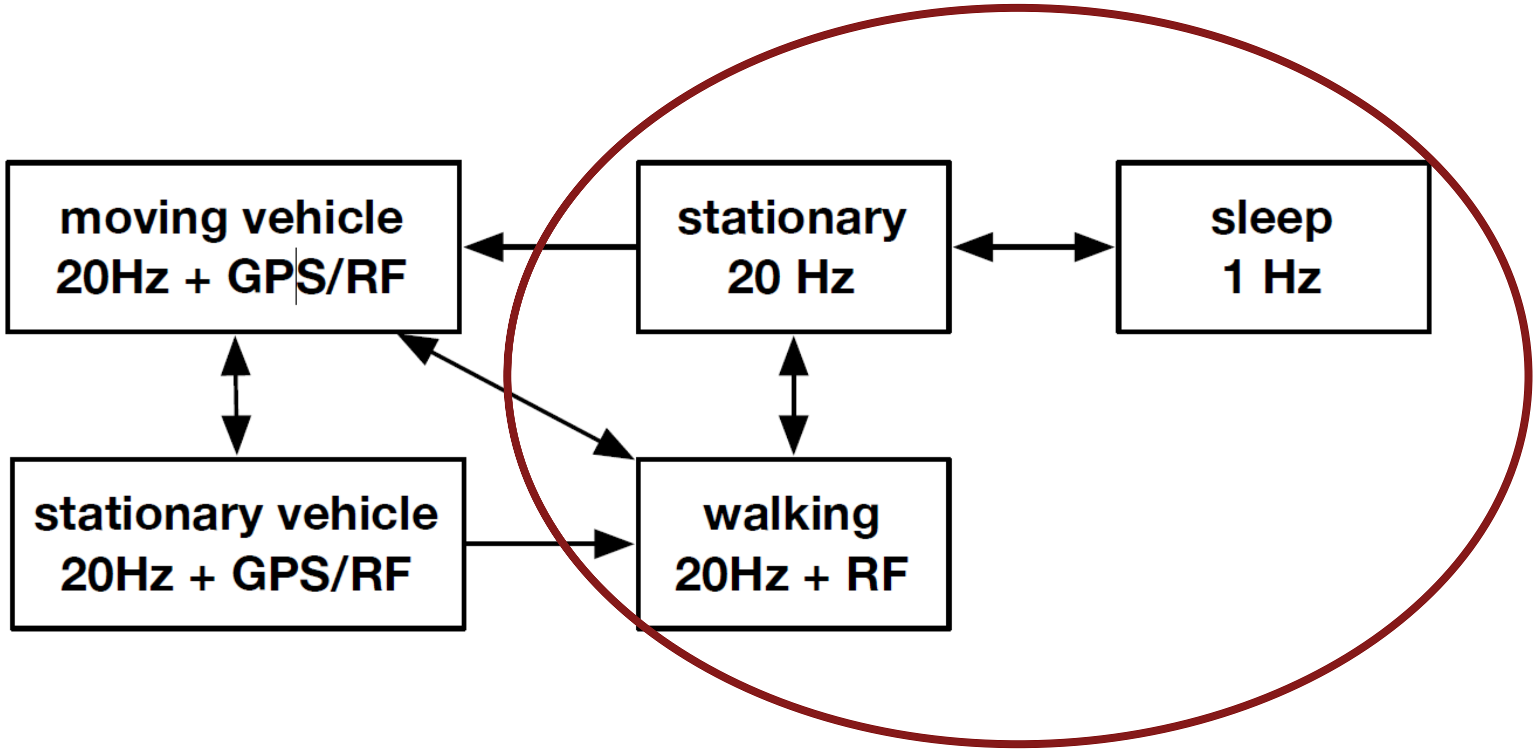
85

Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. **Cooperative transit tracking using smart-phones**.

In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems,*

ACM SenSys 2010.

An example from the literature



Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. **Cooperative transit tracking using smart-phones**. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, ACM SenSys 2010.*

Low-power motion detection

- Detect transitions away from the stationary state (e.g., sitting, standing)
- Sample the accelerometer at 1Hz
- Continuously compute exponentially weighted means and standard deviations of X, Y and Z readings
- If an incoming sample falls outside of three standard deviations of **any** axis, a motion is detected
  - → Increase sampling rate, wake up more energy-hungry sensors

Let’s try it..

On-line mean and std calculation

- Running mean and standard deviation
- Produce incremental results after each sample becomes available

new sample x available:

```
    diff = x - mean
    incr = alpha * diff
    mean = mean + incr
    variance = (1 - alpha) * (variance + diff * incr)
```

[http://nfs-uxsup.csx.cam.ac.uk/~fanf2/hermes/doc/antiforgery/stats.pdf]

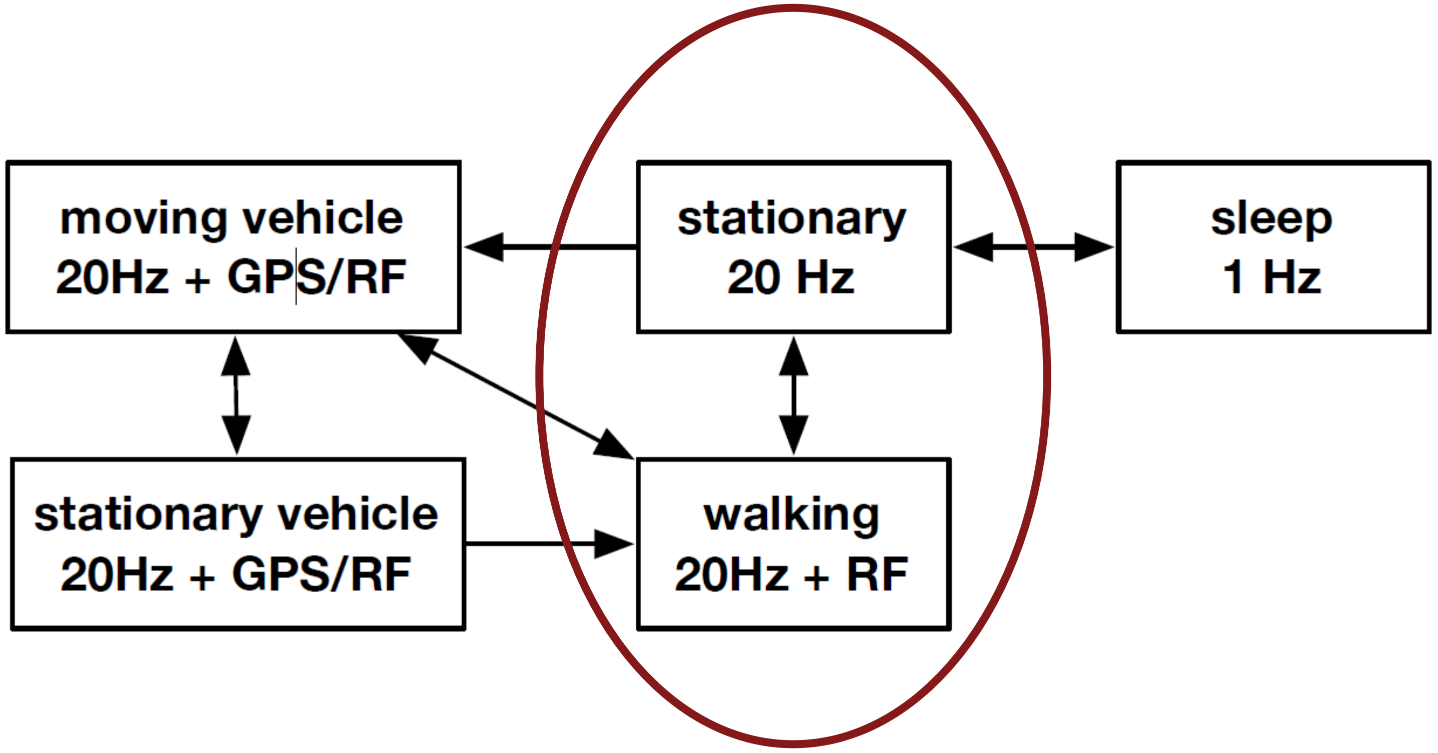
What happens for different alphas?

Low-power motion detection app

```
@Override protected void onCreate(Bundle savedInstanceState) {
        [....]
        senSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        senAccelerometer = senSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        senSensorManager.registerListener(this, senAccelerometer , sensRate);
}

@Override public void onSensorChanged(SensorEvent event) {
        Sensor mySensor = event.sensor;
        if (mySensor.getType() == Sensor.TYPE_ACCELEROMETER) {
                long curTime = System.currentTimeMillis();
                if ((curTime - lastUpdate) > sensRate / 1000.) {
                        lastUpdate = curTime;
                        for( int i = 0; i < axis; ++i){
                                double tsd = 3 * Math.sqrt(var[i]);
                                if ( (event.values[i] > mean[i] + tsd || event.values[i] < mean[i] - tsd ) ){
                                        MOTION DETECTED
                                }
                                double diff = event.values[i] - mean[i];
                                double incr = alpha * diff;
                                mean[i] = mean[i] + incr;
                                var[i] = (1.0 - alpha) * (var[i] + diff * incr);
                        }
                }
        }
    }
```

Data values are not necessarily evenly spaced in time (SensorEvent.timestamp field)

An example from the literature



Arvind Thiagarajan, James Biagioni, Tomas Gerlich, and Jakob Eriksson. **Cooperative transit tracking using smart-phones**. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, ACM SenSys 2010.*

Walking detection

- More complex: **No control over and no knowledge of the orientation or placement of the smartphone**

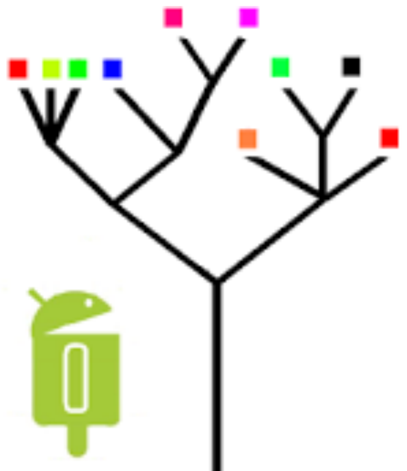
- Increase sampling frequency to 20Hz
- Make raw values orientation-independent by computing the L2-norm (magnitude) of readings

$$\text{Magnitude} = \sqrt{x^2 + y^2 + z^2}$$

- Compute the discrete Fourier transform to detect frequency bands common to walking
- **Binary classification**: walking/not walking
- Decision trees are popular tools for classification:
  - Easy to implement and use
  - Computationally cheap

Decision Tree Learning

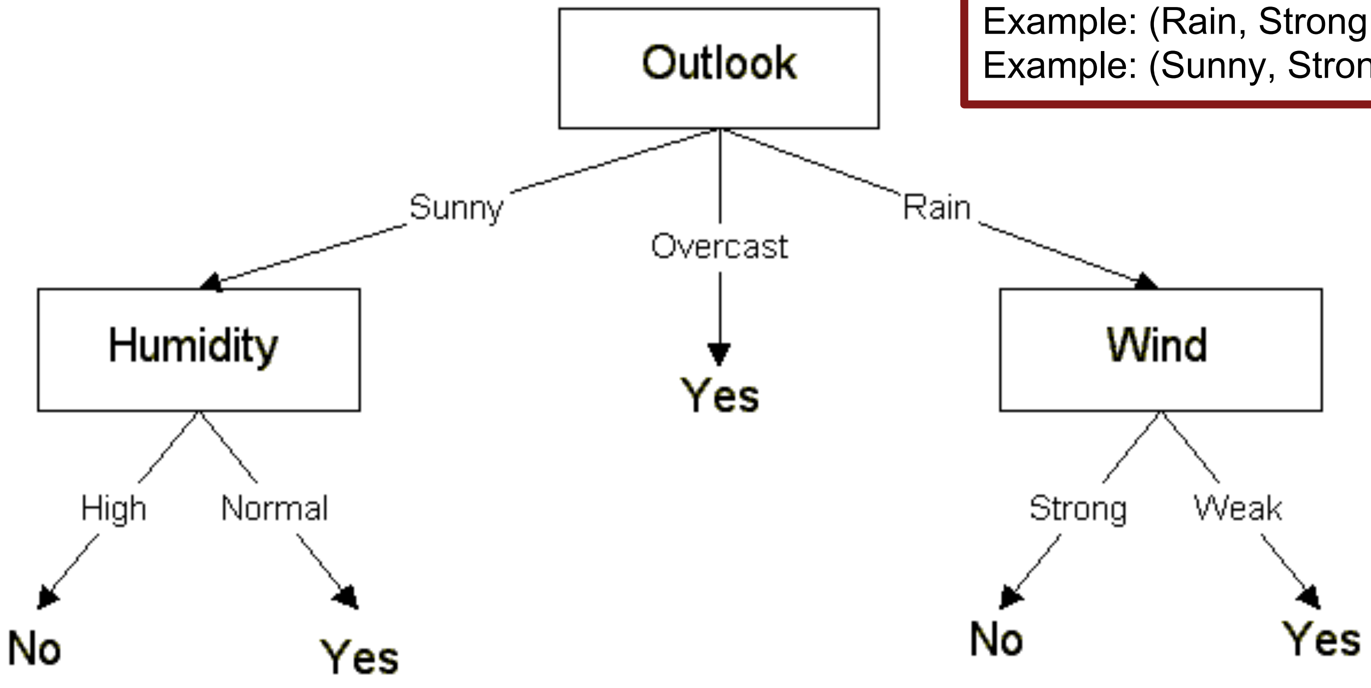
- **Goal**: Classify each item in a dataset into one of predefined set of classes = fixed (known) set of categories
- Given a set of examples with known categories (training dataset), **learn** to assign category to future samples (testing dataset)
- Each example **(instance)** represented by a set of attributes (**features**) that take values in a finite set
- Classification tree:
  - Nodes test features (one branch for each possible value)
  - Leaves specify category



Decision Tree: example

- Features and values:
  - *outlook* {sunny, overcast, rain}
  - *humidity* {high, normal}
  - *windy* {strong, weak}
- Classes: positive instances vs negative instances
  - should we play tennis?

Example: (Rain, Strong, Normal)
Example: (Sunny, Strong, Normal)



Training set example

- Tree built based on a **training set** of **labeled** instances

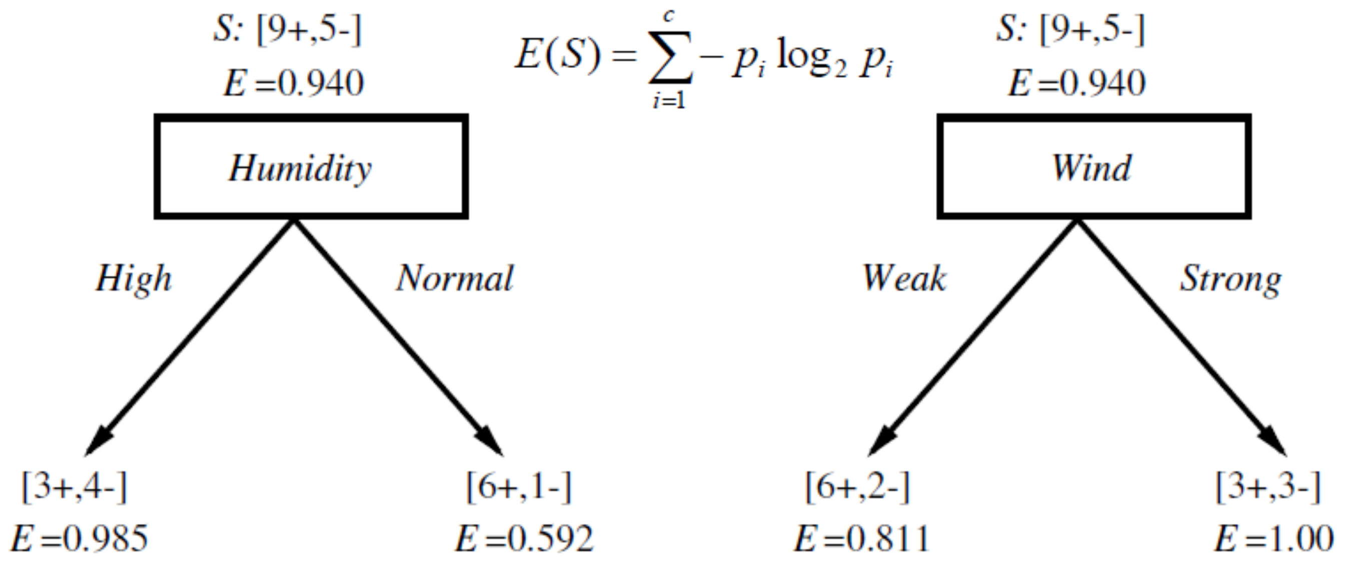
Features

Outlook	**Humidity**	**Wind**	**Play tennis**
Sunny	High	Weak	No
Sunny	High	Strong	No
Overcast	High	Weak	Yes
Rain	High	Weak	Yes
Rain	Normal	Weak	Yes
Overcast	Normal	Strong	Yes
Sunny	High	Weak	No
...	...	...	...

[Full example: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/mlbook/ch3.pdf]

Building a decision tree (ID3)

- Top-down greedy search through the space of possible branches with no backtracking
- Partition data into subsets that contain instances with similar values
- "Best" split based on **information gain** = expected reduction in entropy caused by partitioning the examples with respect to a feature



$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

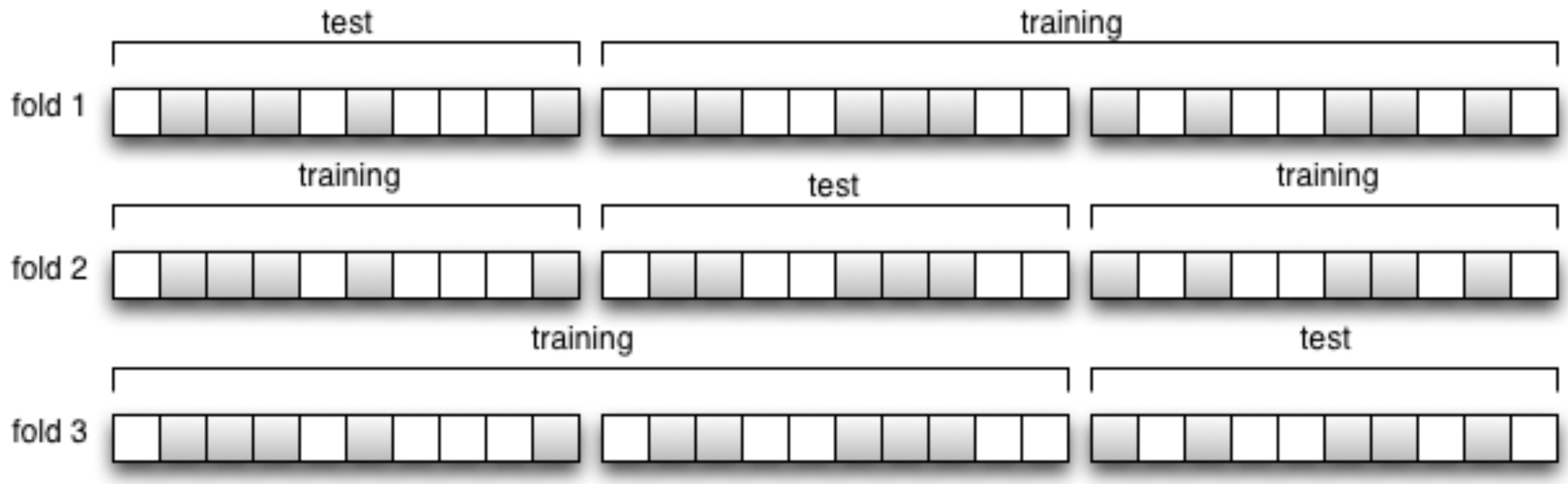
Gain (S, Humidity)
= .940 - (7/14).985 - (7/14).592
= .151

Gain (S, Wind)
= .940 - (8/14).811 - (6/14)1.0
= .048

Prediction performance

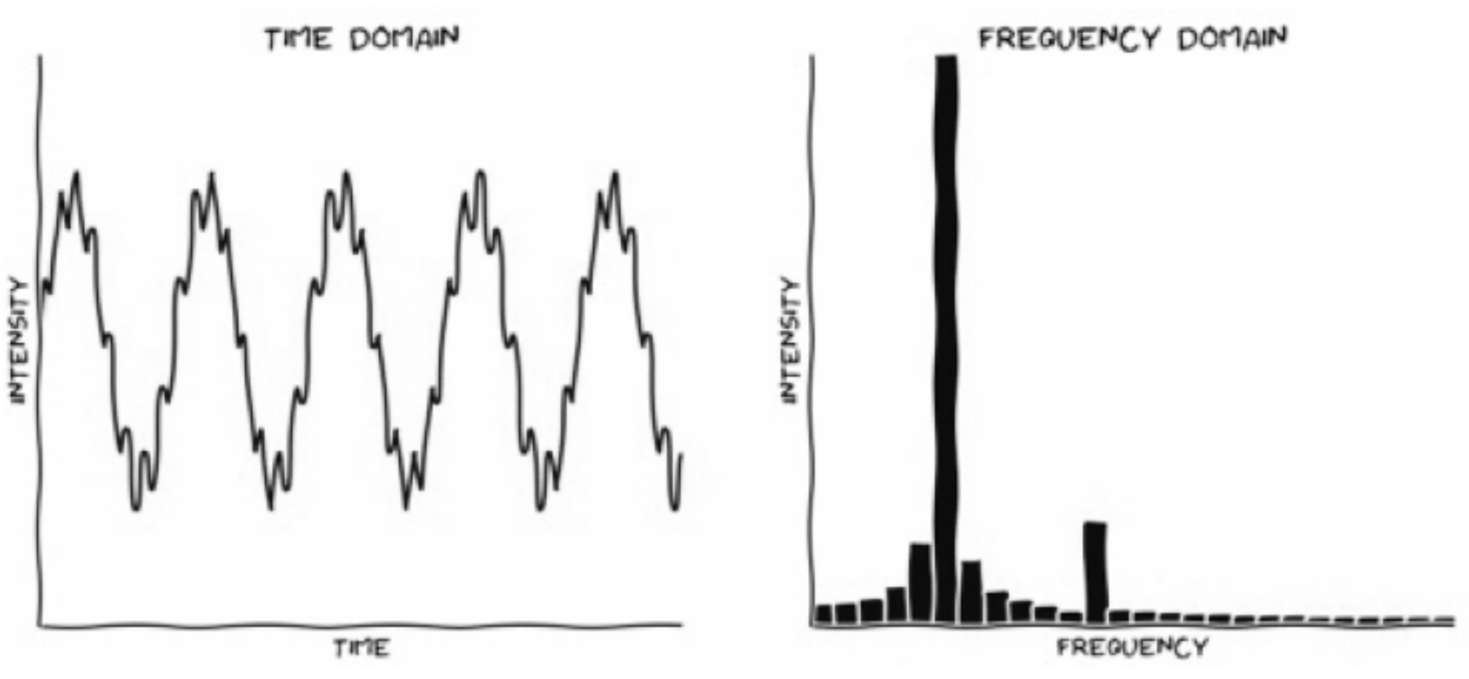
k-fold cross-validation

1. Randomly partition initial samples into k subsets
2. Of the k subsets, k-1 are used for training and the remaining one is used as testing set
3. Validation repeated k times, each subset used exactly once as testing set

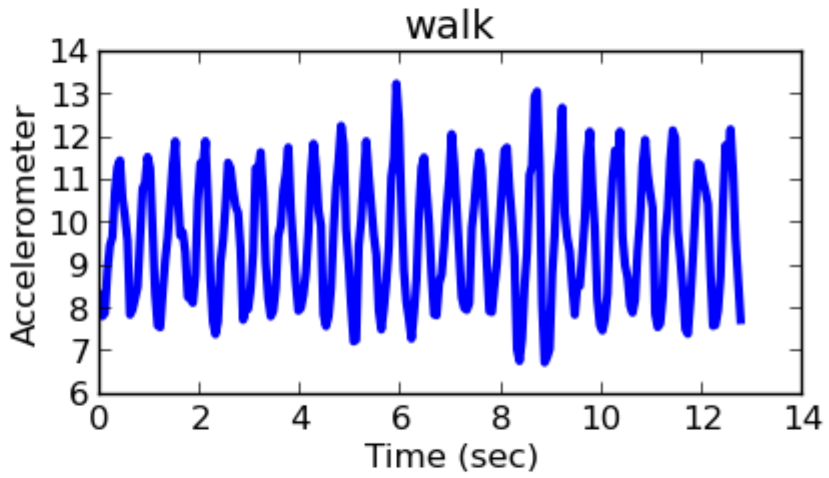


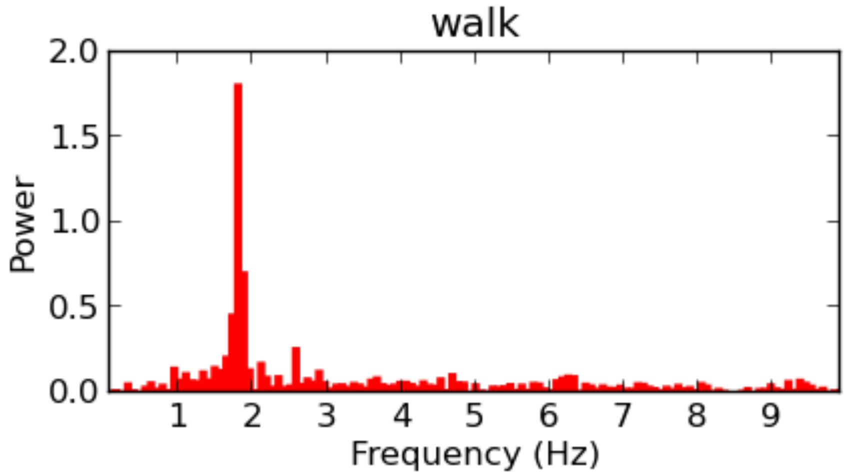
Back to walking detection..

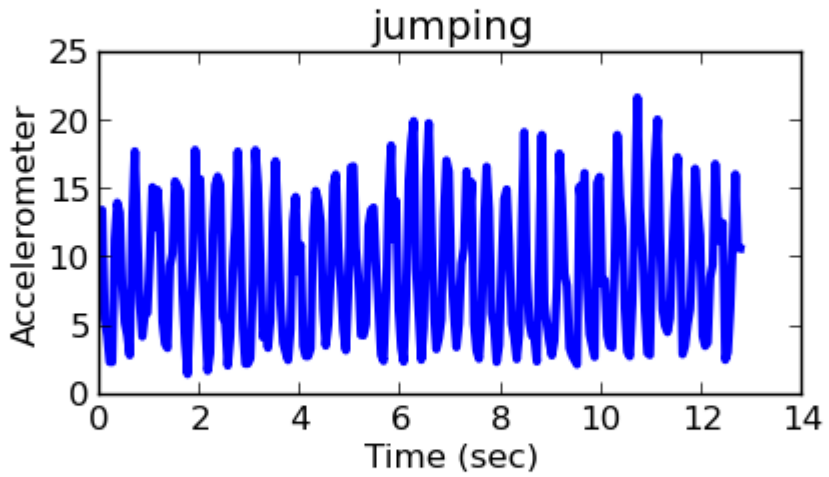
- **Binary classification**: walking/not walking
- Features:
  1. Variance of the sample window
  2. Magnitude of the discrete Fourier transform in frequency bands common to walking (1-3Hz)
  3. Peak frequency power (independent of the walking speed)

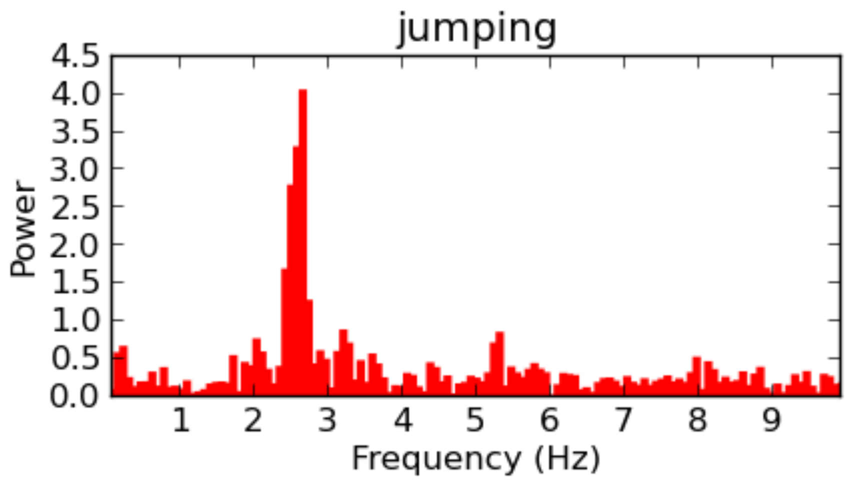


DFT examples

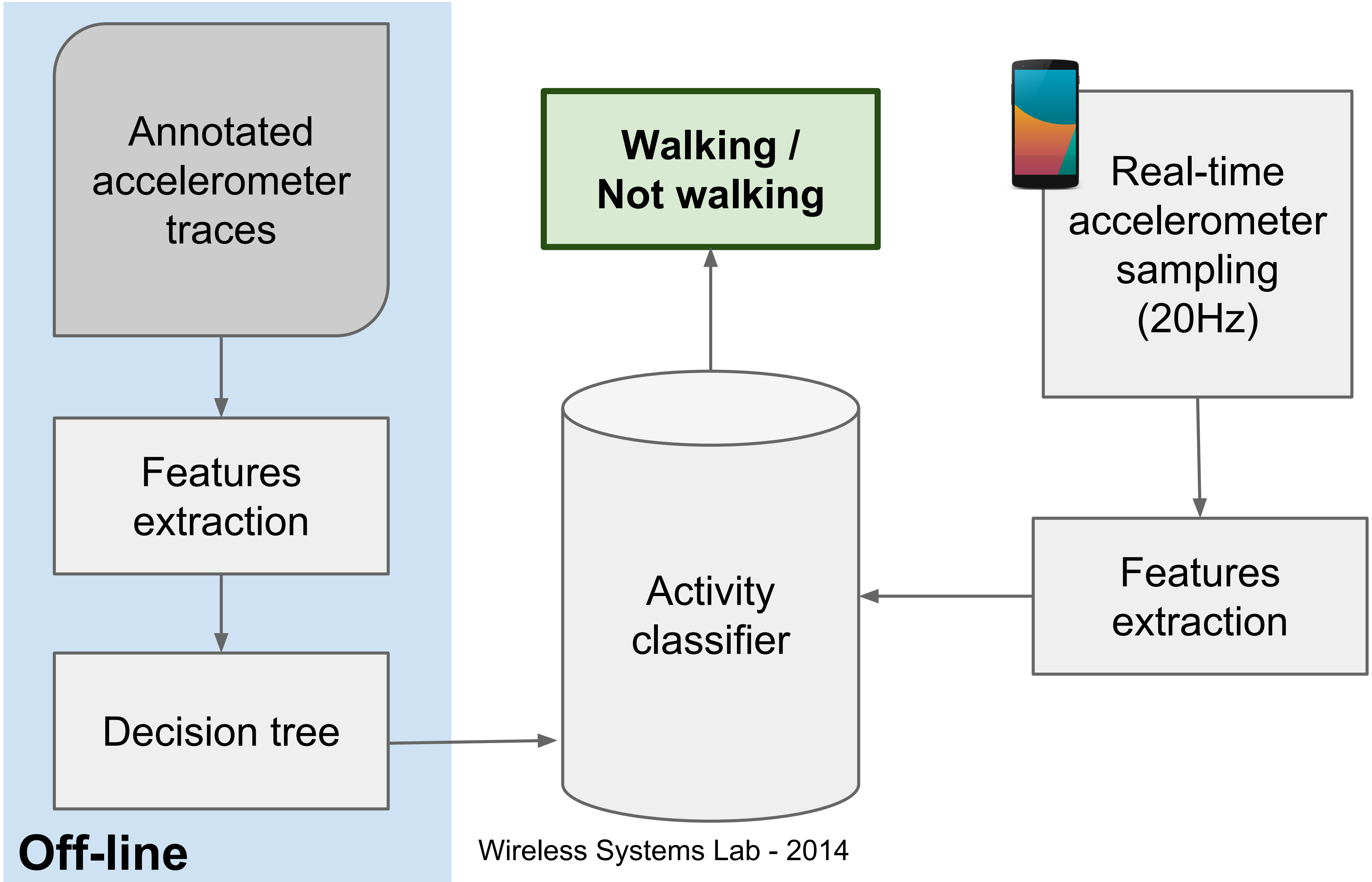








Walking detector: architecture



Walking detector performance

- Training set:
  - “walk”: 5 volunteers walking while varying location of the phone
  - “not walk”: bus, train, car and bike rides; stationary users; waving phone around
- 10-fold cross validation
- Window size = 256 samples
- Classification every 1.5 seconds (32 samples @ 20Hz)

	Walk	non-Walk	Walk	non-Walk
Walk	92%	8%	97.5%	2.5%
Non-Walk	0.4%	99.6%	0.1%	99.9%
	Without Peak Power		With Peak Power	

Hands on!

Build a walking classifier (off-line)

1. Read accelerometer traces **http://wwwusers.di.uniroma1.it/~spenza/files/labWireless2014/accelerometer-traces.tar.bz2**
2. Every 32 samples
   a. Consider a sliding window (size w = 256 samples)
   b. Compute L2-norm
   c. Compute the Discrete Fourier transform (**numpy.fft**)
   d. Store features:
      - Variance of the sample window
      - Peak power frequency
      - Power of the DFT coefficient in the 1-3Hz range
3. Build classifier (**sklearn.tree.DecisionTreeClassifier**)
4. Test performance with 10-fold cross validation

Compute features: DFT

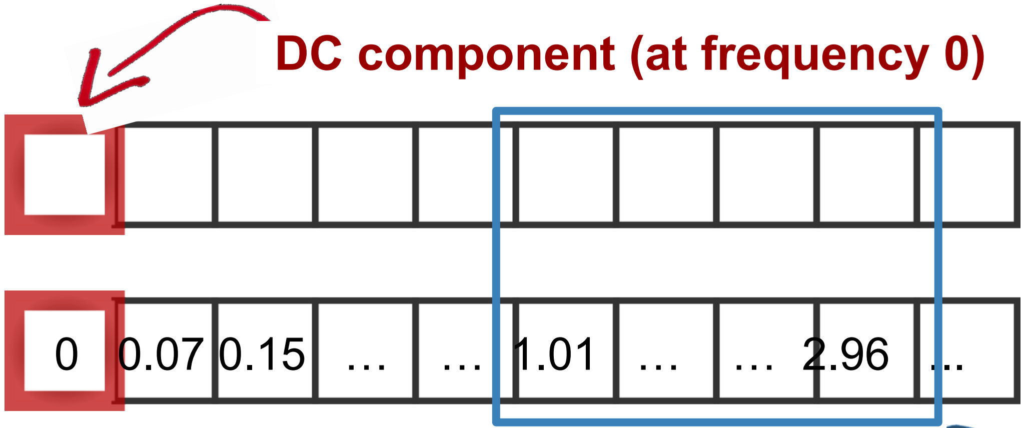
DFT definition

$$A_k = \sum_{m=0}^{n-1} a_m \exp\left\{-2\pi i \frac{mk}{n}\right\} \qquad k = 0, \ldots, n-1.$$

Inverse of sampling rate

```
fft_x = numpy.fft.fft(x)
l = len(fft_x)

freq = numpy.fft.fftfreq(l, 1.0 / 20 )                    # Matching vector of frequencies
fft_x_shifted = numpy.fft.fftshift(fft_x)                  # Shift DC component
half_l = numpy.ceil(l/2.0)
fft_x_half = numpy.abs( (2.0 / n) * fft_x[:half_l] )       # Fold negative frequencies and scale
freq_half = freq[:half_l]
```

DC component (at frequency 0)



fft_x_half: amplitude of the FFT at positive frequencies

freq_half: frequency bins (Hz)

Frequency range 1-3Hz

Compute features

# Variance of the sample window (time domain)

variance = $\sigma^2 = \frac{\sum (X-\mu)^2}{N}$

# Peak frequency: frequency at which the amplitude is max (excluding DC component)

pf_index = …

# Amplitude of the DFT in the 1-3 Hz range

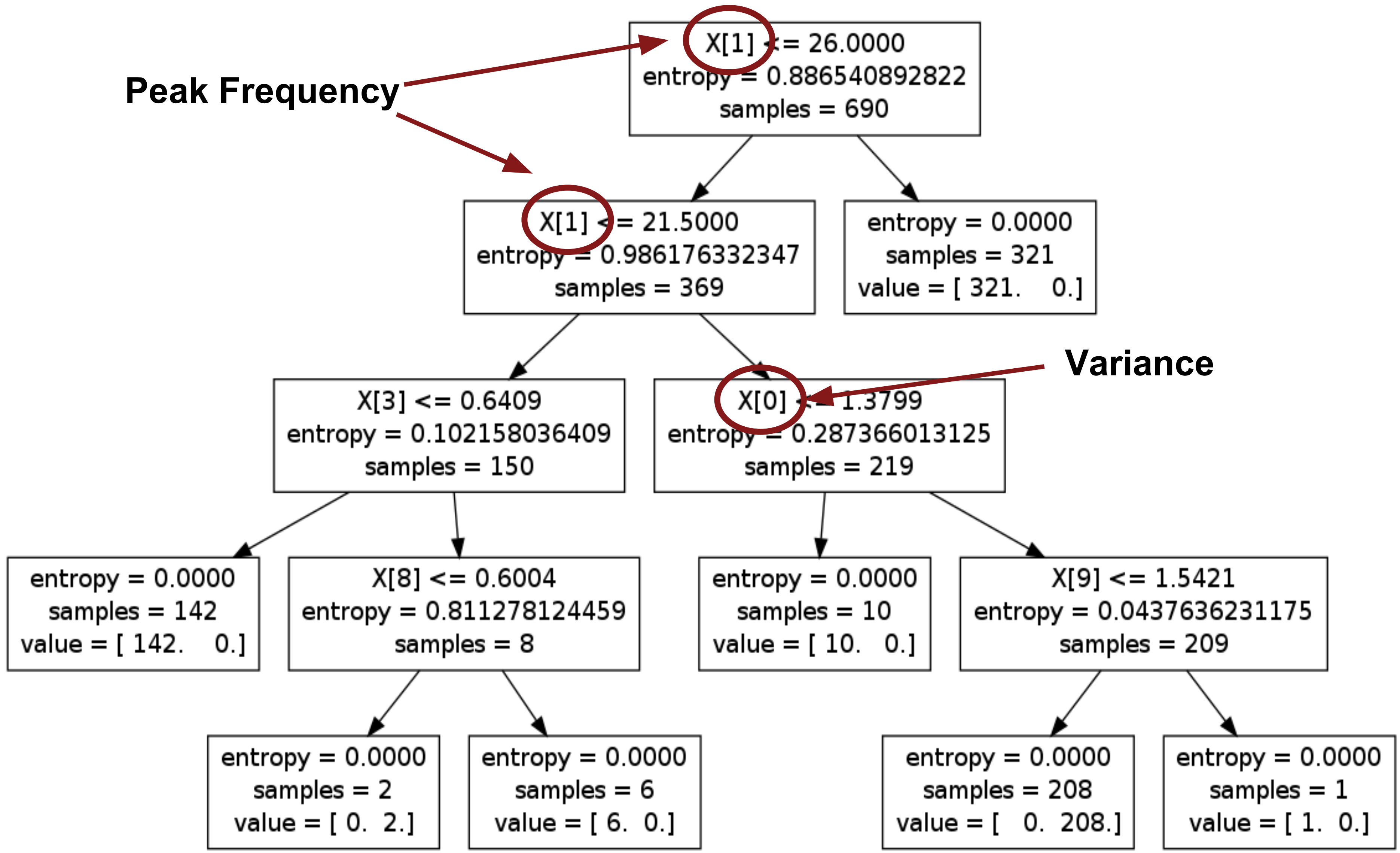
freqs = ….

return [variance, pf_index] + freqs

How to build the classification tree

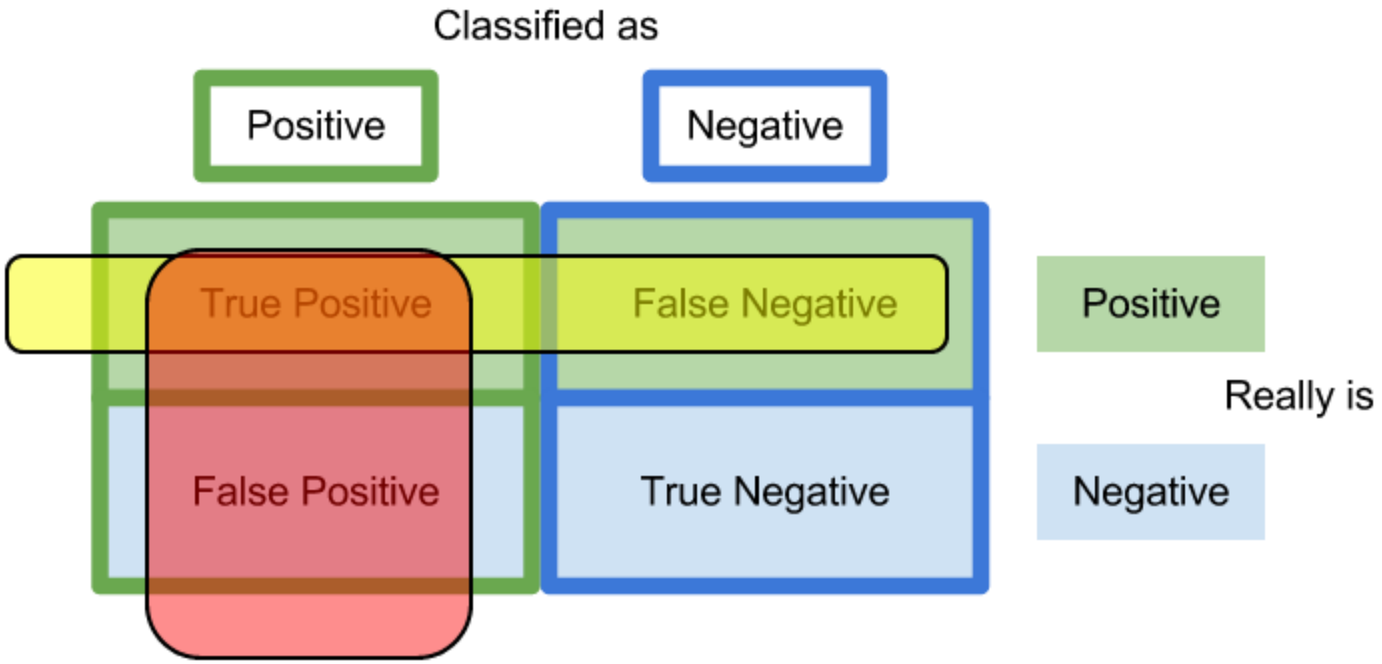
```
from sklearn import tree

samples = list of computed features
classes = classification of each sample (walking/not walking)

clf = tree.DecisionTreeClassifier(criterion='entropy', random_state=0)
clf = clf.fit(samples, classes)

tree.export_graphviz(clf, out_file='trees/tree.dot')
os.system("dot -Tpng trees/tree.dot -o trees/tree.png" )
```

Resulting decision tree



Classification performance

- **Precision**: ratio tp / (tp + fp). Intuitively, ability of not to label as positive a sample that is negative.
- **Recall**: ratio tp / (tp + fn). Intuitively, ability to find all the positive samples.



Measuring performance

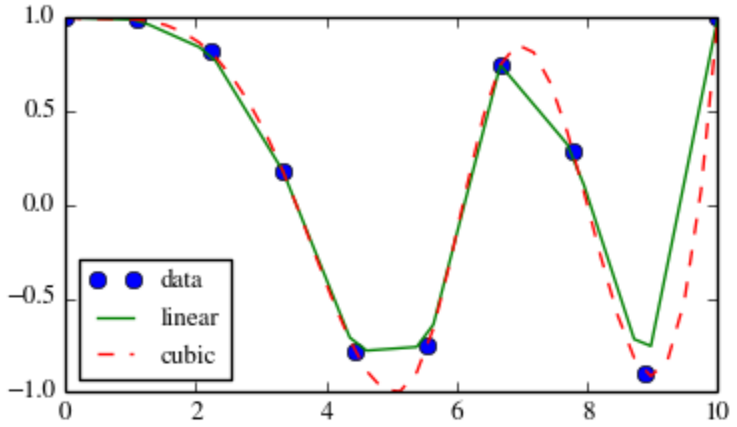
```
from sklearn import cross_validation
scores = cross_validation.cross_val_score(clf, samples, classes, cv=10,
scoring="recall")
print("Recall: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
scores = cross_validation.cross_val_score(clf, samples, classes, cv=10,
scoring="precision")
print("Precision: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

	With peak power	**Without peak power**
Recall	0.98 (+/- 0.08)	0.81 (+/- 0.37)
Precision	0.98 (+/- 0.05)	0.90 (+/- 0.29)

Data pre-processing

- Accelerometer data are not generally evenly spaced in time (check SensorEvent.timestamp field)
- **DFT** requires a finite list of **equally spaced** samples of a function

➔ **Interpolate** accelerometer traces



```
from scipy.interpolate import interp1d
f = interp1d(timestamps, accelerometer, kind='cubic')
new_timestamps = np.arange(0, timestamps[-1], s_p)
es_accelerometer = f(new_timestamps)
```

Using the classifier online

1. Convert the decision tree to a sequence of rules and implement them in the app

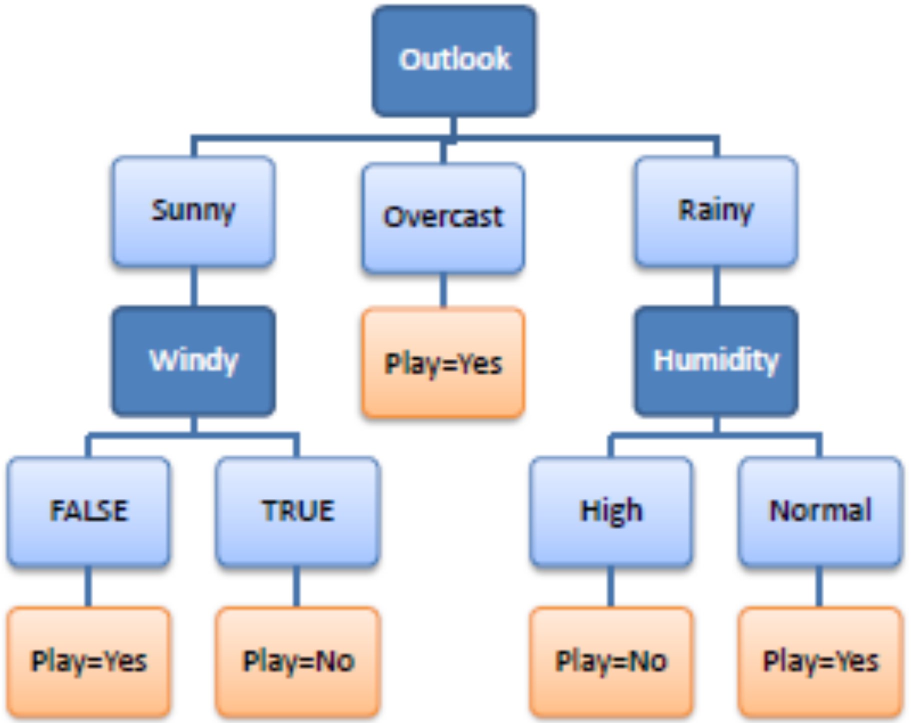
R_1: IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

R_2: IF (Outlook=Sunny) AND (Windy=TRUE) THEN Play=No

R_3: IF (Outlook=Overcast) THEN Play=Yes

R_4: IF (Outlook=Rainy) AND (Humidity=High) THEN Play=No

R_5: IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=Yes



Using the classifier online

1. Convert the decision tree to a sequence of rules and implement them in the app
2. App samples accelerometer @ 20Hz
3. Performs classification every 32 samples
4. Computes features based on the last 256 samples:
   a. Variance of the sample window
   b. Peak power frequency
   c. Power of the DFT coefficient in the 1-3Hz range
5. Feed features to the classifier
6. Output classification (walking/not walking)

Homework

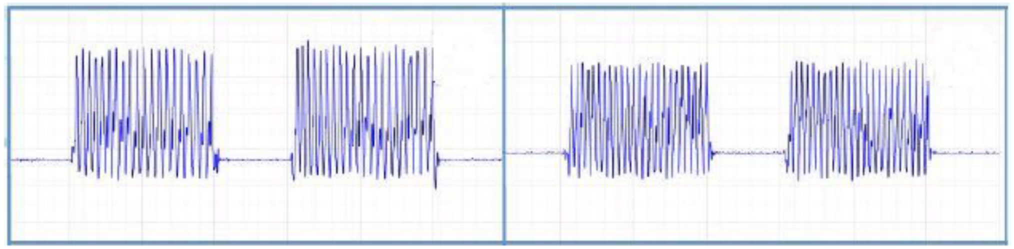
Write an app to collect your own accelerometer data traces

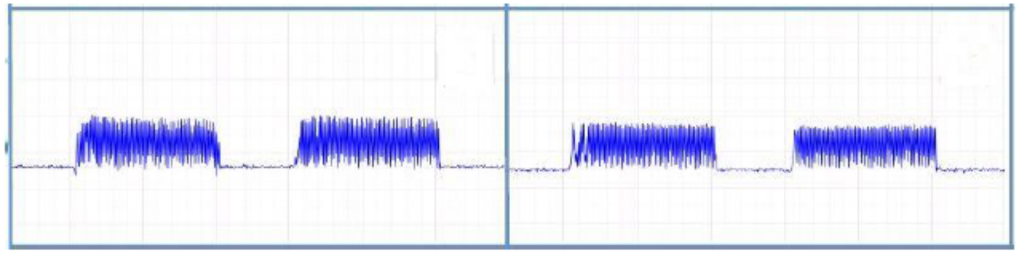
1. Read accelerometer @20Hz
2. Use the External Storage to store collected data in a file. Format:
   timestamp, acc_x, acc_y, acc_z
3. Collect training data:
   a. Perform different activities (e.g., walking, dancing, standing on a bus, …)
   b. Label traces with activity

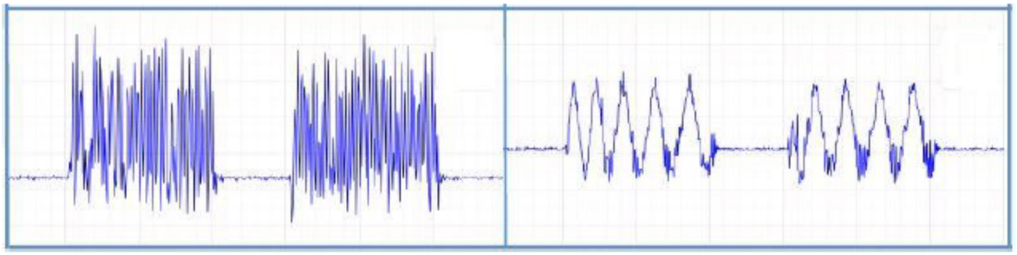
Privacy concerns

- Several commonly-available sensors do not require explicit permission for data reading
- Can be done by apps silently
- Privacy concerns
- Example: accelerometer:
  - Can be used to identify user activity
  - They have **unique fingerprints** (see next slides)

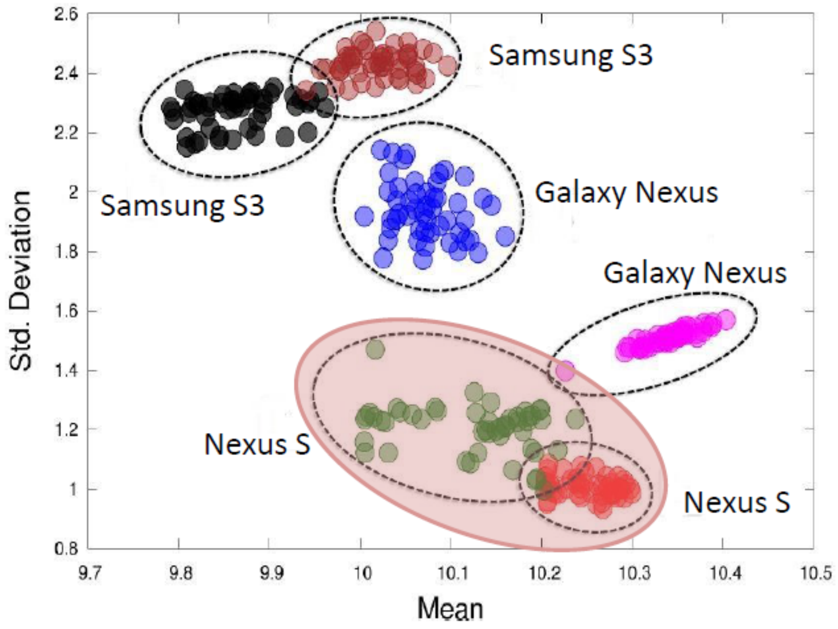
Accelerometers have fingerprints


Accelerometer chips of Samsung Galaxy S3

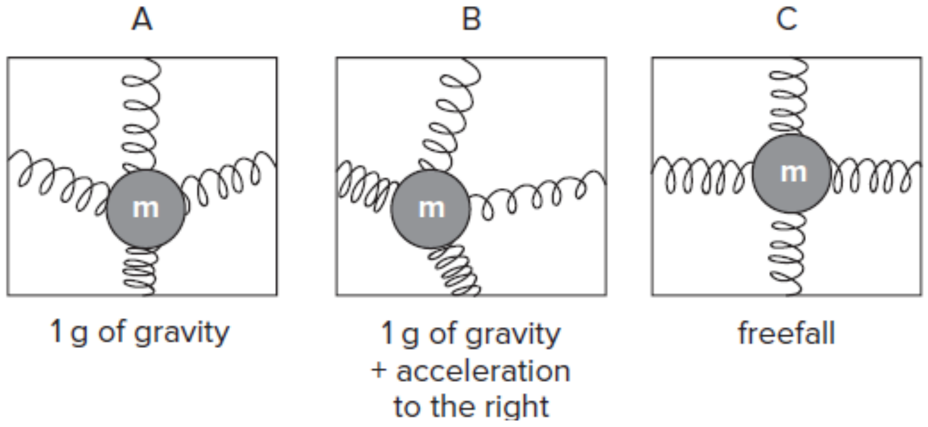

Accelerometer chips of Nexus S


Accelerometer chips of Samsung Galaxy Nexus

Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury and Srihari Nelakuditi. **AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable.** In proceedings of NDSS 2014. [http://www.internetsociety.org/sites/default/files/03_2_1.pdf]

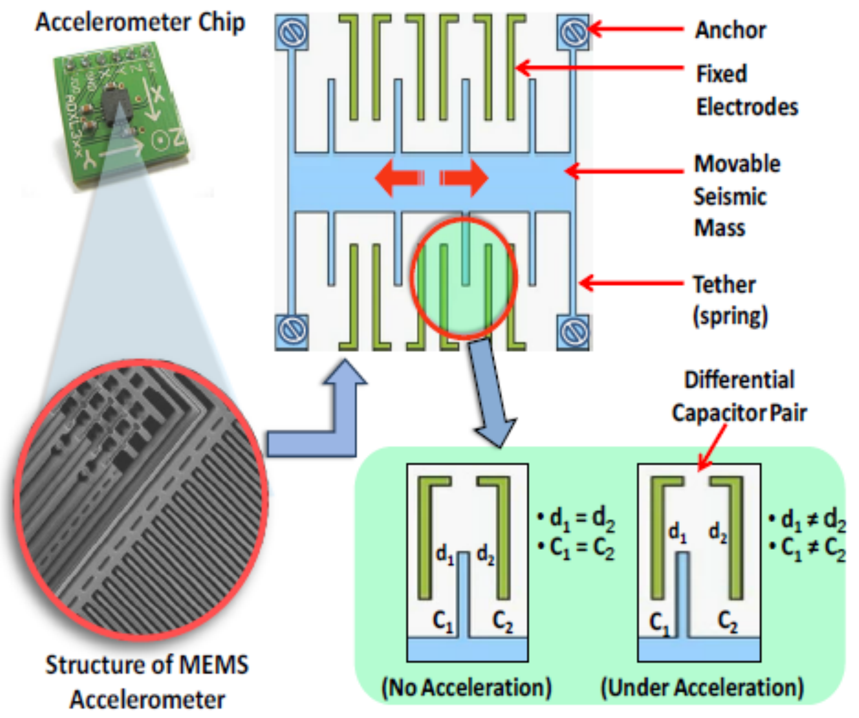


Hardware imperfections



Small gaps between structural parts can change the **absolute value** of the capacitance

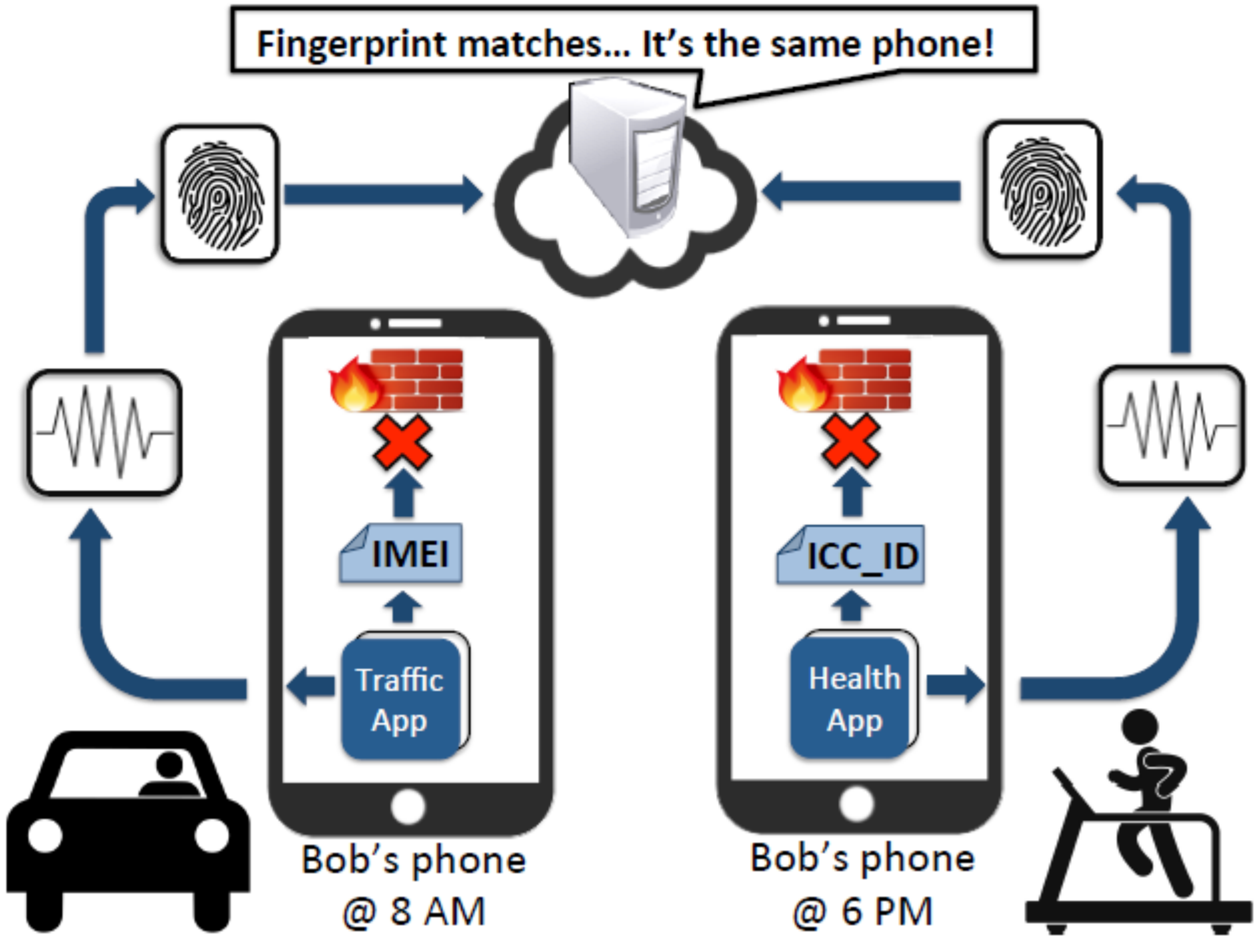
Target applications for smartphones are marginally affected, as they primary depends on the **relative change** in accelerometer readings



$$C = \frac{\varepsilon A}{d}$$

http://www.instrumentationtoday.com/mems-accelerometer/2011/08/

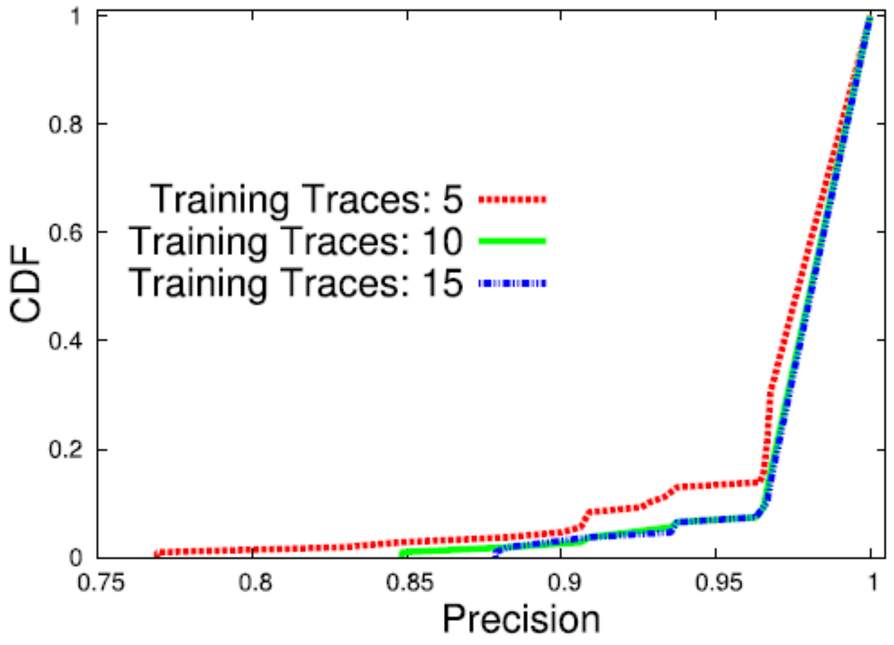
Recognize user based on accelerometer hw

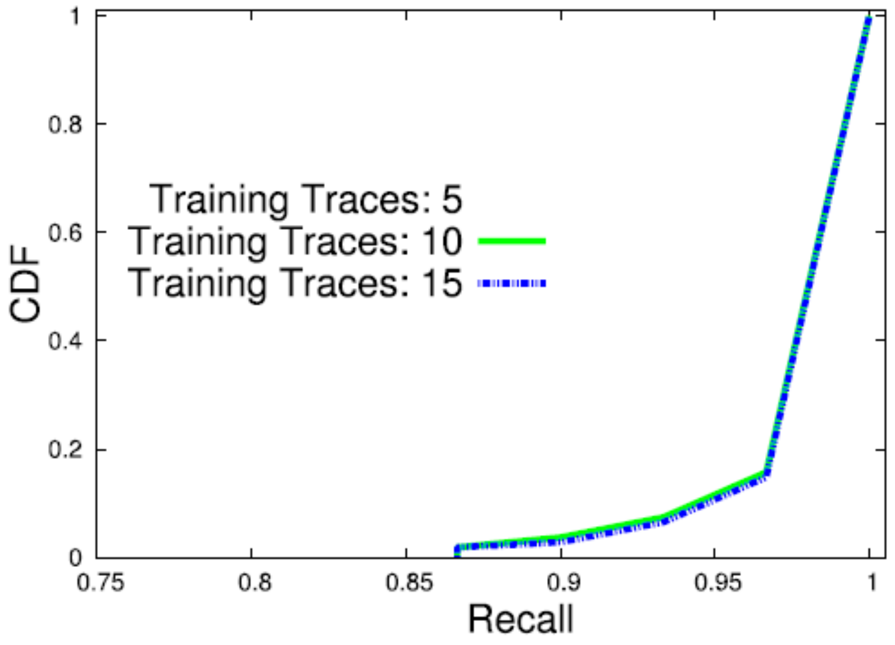


Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury and Srihari Nelakuditi. **AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable.** In proceedings of NDSS 2014. [http://www.internetsociety.org/sites/default/files/03_2_1.pdf]

Large scale exploration

- 107 stand-alone chips, smartphones and tablets
- 36 time domain and frequency domain features
- 30 seconds of acc.data enough to model fingerprint





[http://www.internetsociety.org/sites/default/files/03_2_1.pdf]

average precision & recall > 99%