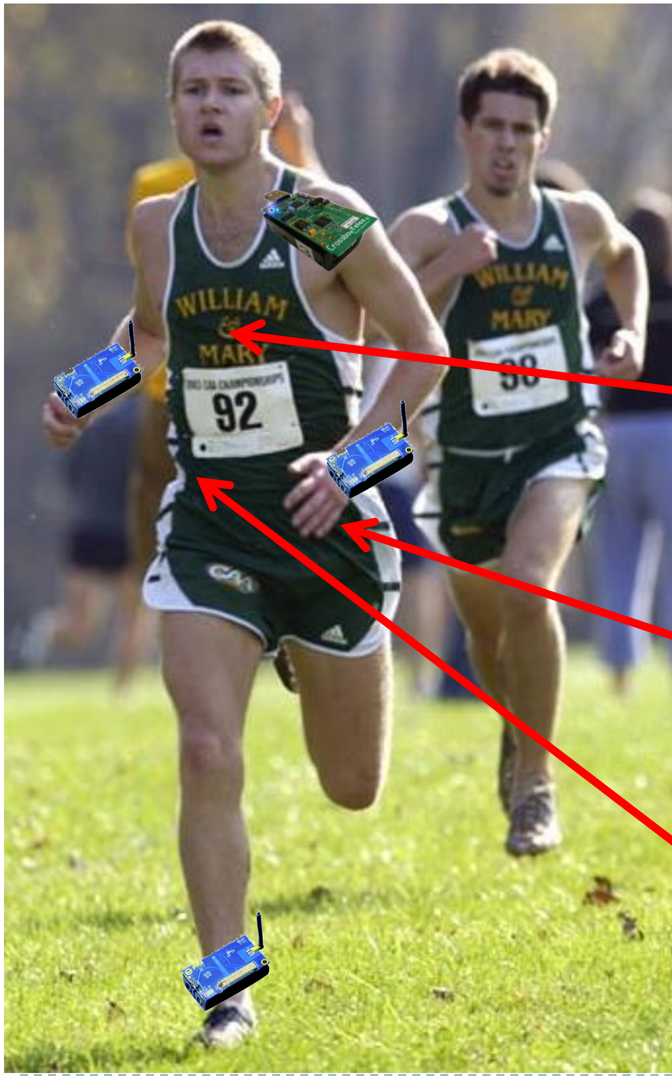
Human Activity Recognition

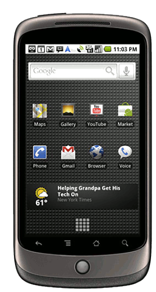
Personal Sensing Applications



- Body Sensor Networks
  - Athletic Performance
  - Health Care
  - **Activity Recognition**

Heart Rate Monitor

Pulse Oximeter

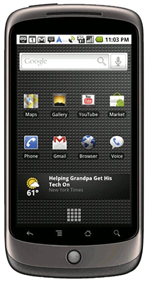


Mobile Phone Aggregator

A Practical Solution to Activity Recognition

- Portable
- Entirely user controlled
- Computationally lightweight
- Accurate

On-Body Sensors
+Sensing Accuracy
+Energy Efficiency



Phone
+User Interface
+Computational Power
+Additional Sensors

Application requirement

- Activity recognition
- Data comes from different sensors
- Classify typical daily activities, postures, and environment
- Classification Categories:

Environment	Indoors, Outdoors
Posture	Cycling, Lying Down, Sitting, Standing, Walking
Activity	Cleaning, Cycling, Driving, Eating, Meeting, Reading, Walking, Watching TV, Working

Challenges to Practical Activity Recognition

- User-friendly
  - Hardware configuration
    - Portable sensors , easy to wear
  - Software configuration
    - Intuitive interface, adding, removing, config. sensors
- Accurate classification
  - Classify difficult activities in the presence of dynamics
    - Noisy env., orientation of sensors
- Efficient classification
  - Computation and energy efficiency
- Less reliance on ground truth
  - Labeling sensor data is invasive

PBN: Practical Body Networking

Tools

- TinyOS-based motes + Android phone

Goals

- Lightweight activity recognition appropriate for motes and phones
- Retraining detection to reduce invasiveness
- Identify redundant sensors to reduce training costs
- Classify difficult activities with nearly 90% accuracy

PBN system

- Crossbow IRIS on body sensor motes
- TelosB base station
  - Connected with HTC smartphone

TinyOS sensing support

- Implement sensing application in TiniOS for motes
- Runtime configuration of active sensors, sampling rate, local aggregation
- Communication scheme =>base station=>phone

Android kernel support for USB

- Prepare for external USB
- Driver installation

Hardware support

- Ext. battery power for the motes

TinyOS support on Android

- Enable TinyOS and Android communication

Android App

- User friendly front end
- Easy configuration
- Runtime deployment
- Labelling
- User control for both phone and motes
- Receives feedback if retraining is needed

Android App

- Sensor configuration

Easy config for phone and motes

Add/remove sensors

Adjust sampling rate, local aggregation interval

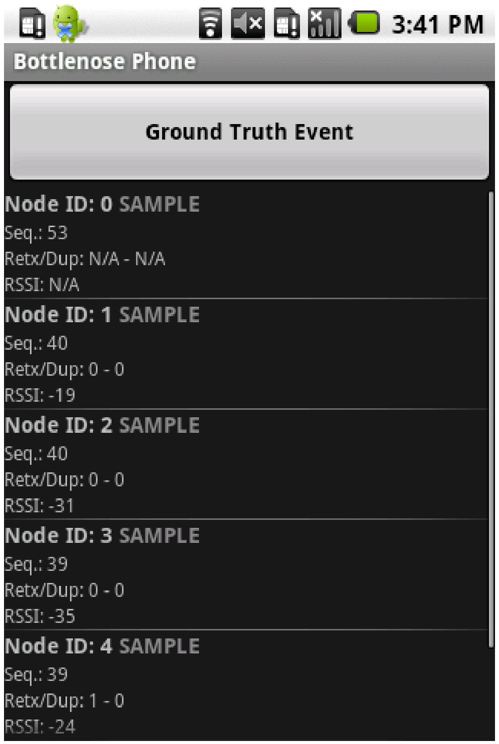
Save on XML

- Runtime control

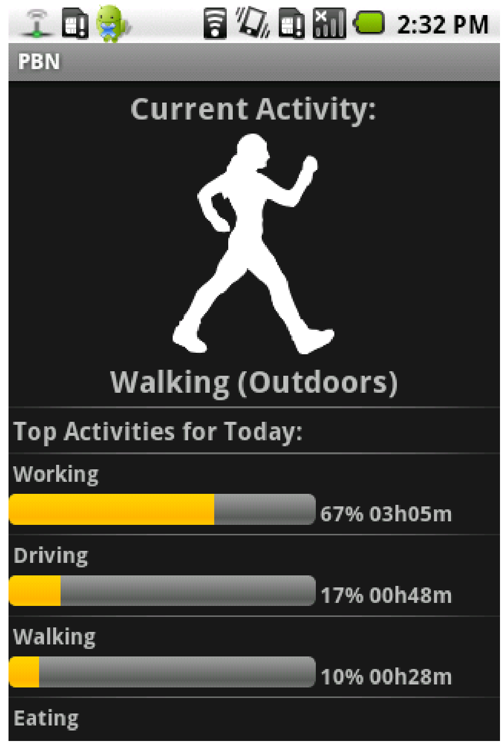
User is able to start/stop data sampling and activity recog.

Retraining => enter current activity

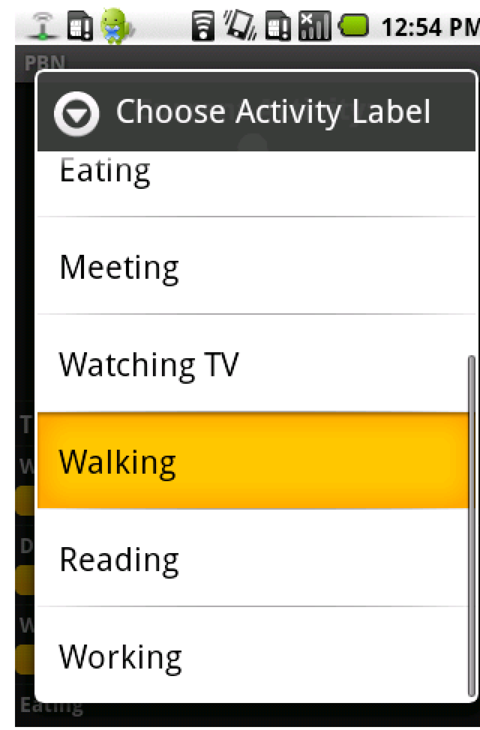
Software: Android Application



Sensor Configuration



Runtime Control and Feedback

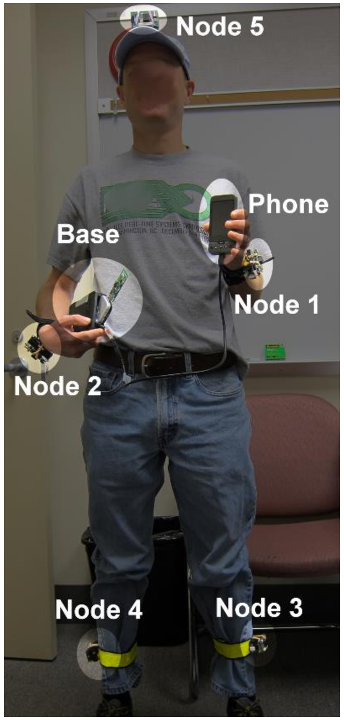


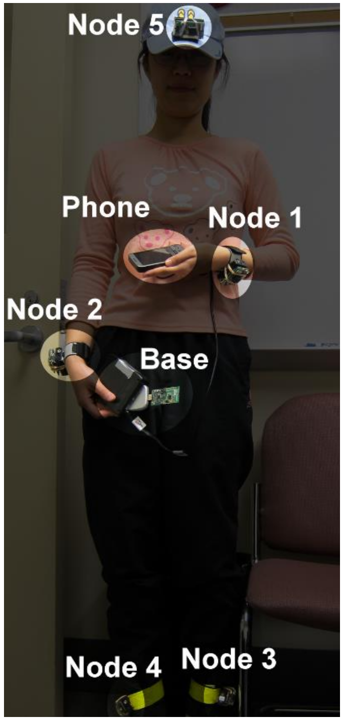
Ground Truth Logging

Data Collection Setup

- 2 subjects, 2 weeks
- Android Phone
  - 3-axis accelerometer, WiFi/GPS Localization
- 5 IRIS Sensor Motes
  - 2-axis accelerometer, light, temperature, acoustic, RSSI

Node ID	Location
0	BS/Phone
1	L.Wrist
2	R.Wrist
3	L.Ankle
4	R.Ankle
5	Head

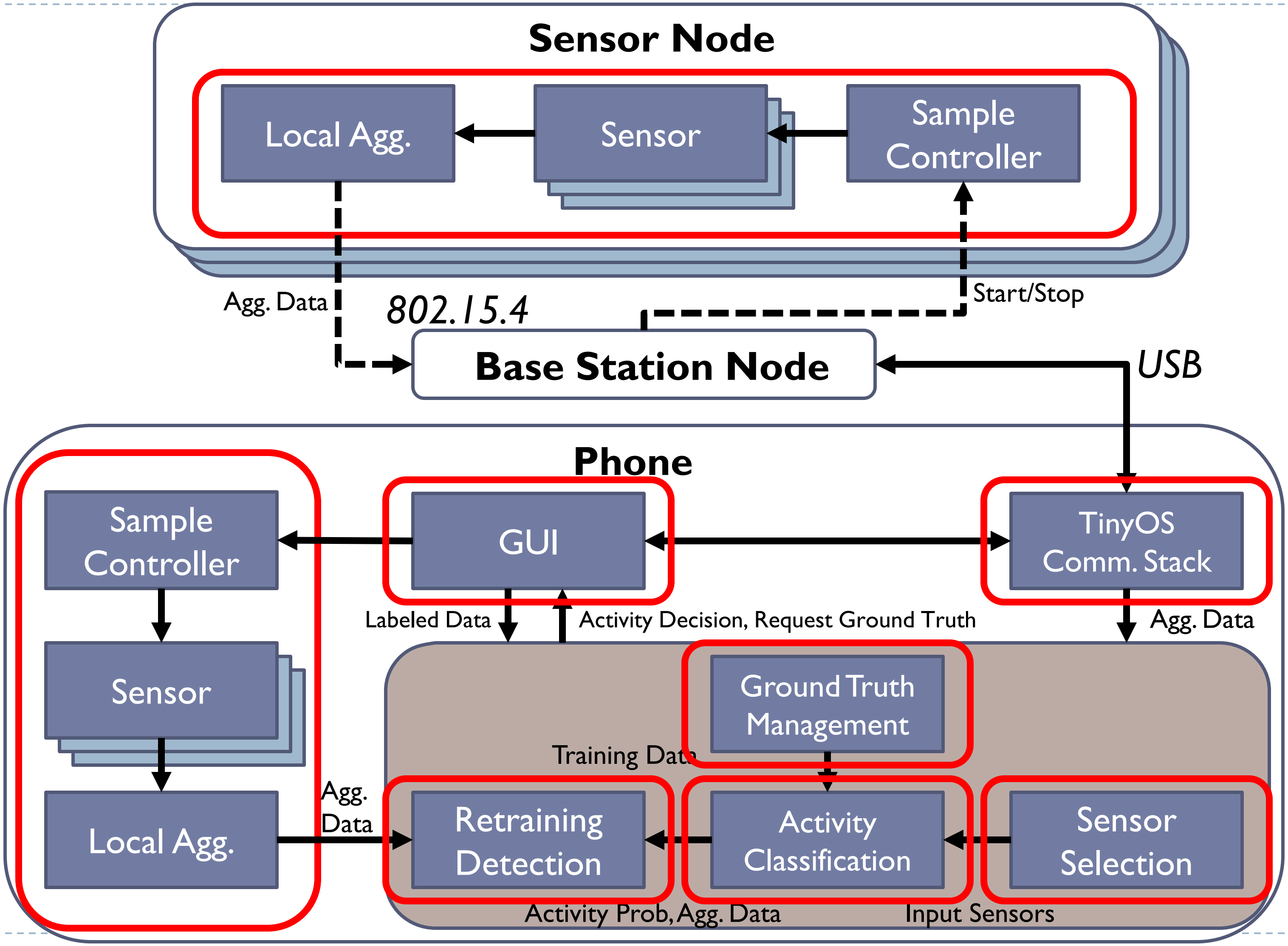




Node	ID	Location	Sensors
Phone	0	R. Waist	3-Axis Acc., GPS/WiFi (velocity)
IRIS	1	L. Wrist	2-Axis Acc., Mic., Light, Temp.
IRIS	2	R. Wrist	2-Axis Acc., Mic., Light, Temp.
IRIS	3	L. Ankle	2-Axis Acc., Mic., Light, Temp.
IRIS	4	R. Ankle	2-Axis Acc., Mic., Light, Temp..
IRIS	5	Head	2-Axis Acc., Mic., Light, Temp.

Signal strength

PBN Architecture



PBN Architecture

- Phone and mote sensors sample data

Aggregate => single packet

- Fed to classification system

AdaBoost => classifier , each activity training

Two minutes period

Updated using retraining

Sensor selection

AdaBoost Activity Recognition

- Ensemble Learning: AdaBoost.M2 (Freund, JCSS '97)\\\\\\\\\\\\
  - Lightweight and accurate
  - Maximizes training accuracy for all activities
  - Many classifiers (HMM) are more demanding

- Iteratively train an ensemble of weak classifiers
  - Training observations are weighted by misclassifications
  - At each iteration:
    - Train Naïve Bayes classifiers for each sensor
    - Choose the classifier with the least weighted error
    - Update weighted observations

- The ensemble makes decisions based on the weighted decisions of each weak classifier

AdaBoost

Ensemble classifier
Weak classifier
Combined to make a single classifier

Using Algorithm 1, we describe AdaBoost training. We define a set of activities $A = \{a_1, \ldots, a_a\}$, sensors $S = \{s_1, \ldots, s_m\}$, and observation vectors O_j for each sensor $s_j \in S$, where each sensor has n training observations. The training output is an ensemble of weak classifiers $H = \{h_1, \ldots, h_T\}$, where $h_t \in H$ represents the weak classifier

Initialize the weight vector D

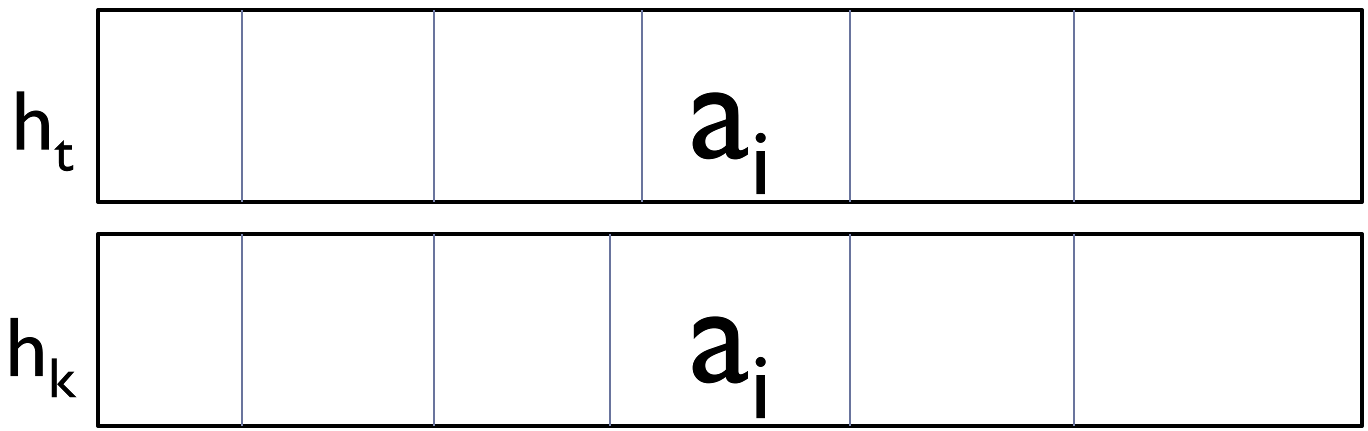
Algorithm 1 AdaBoost Training

Input: Max iterations T, training obs. vector O_j for each sensor $s_j \in S$, obs. ground truth labels

Output: Set of weak classifiers H

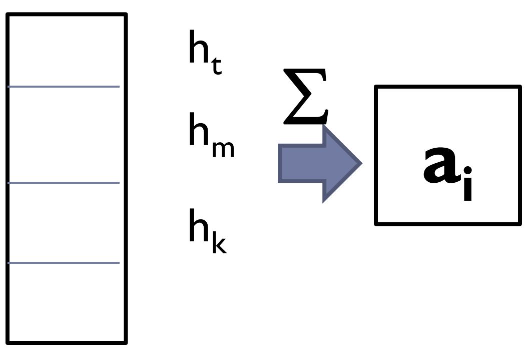
Initialize observation weights D_1 to $1/n$ for all obs.
for $t = 1$ to T **do**
    for sensor $s_j \in S$ **do**
        Train weak classifier $h_{t,j}$ using obs. O_j, weights D_t
        Get weighted error $\varepsilon_{t,j}$ for $h_{t,j}$ using labels [8]
    end for
    Add the $h_{t,j}$ with least error ε_t to H by choosing $h_{t,j}$ with least error ε_t
    Set D_{t+1} using D_t, misclassifications made by h_t [8]
end for

Final outcome of AdaBoost



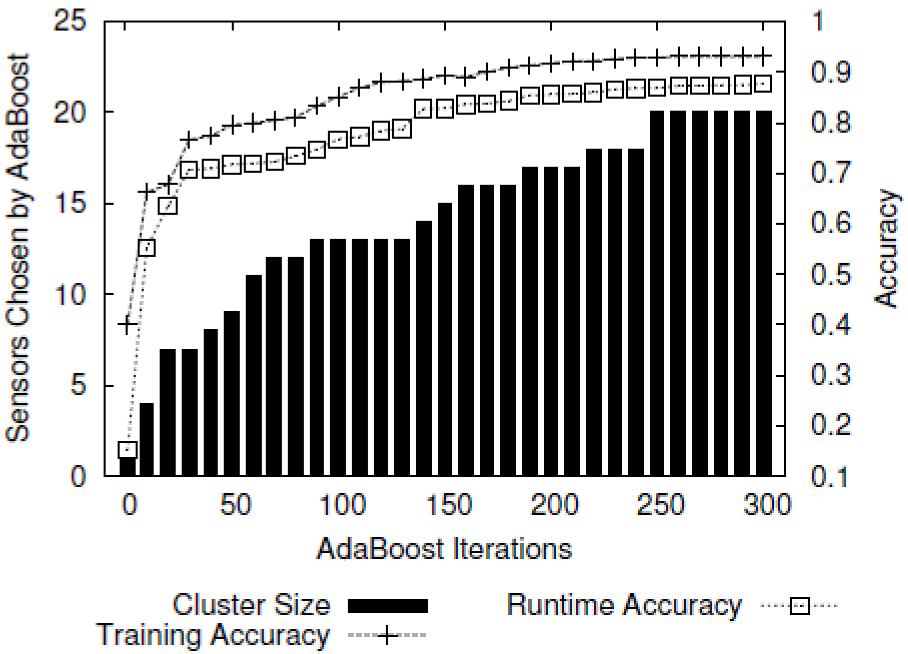
Given a observation o, weak classifier h_t returns a vector [0,1]

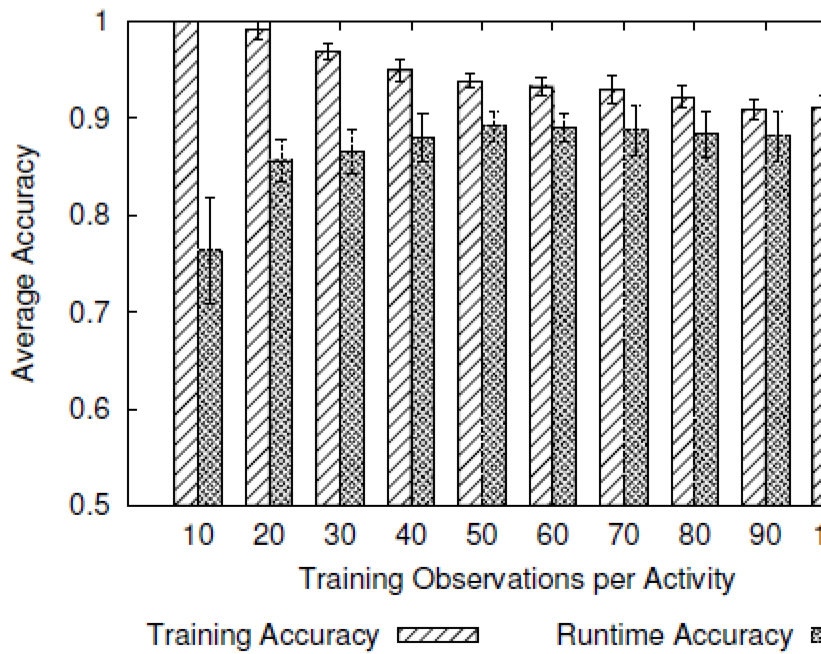
Activity $\mathbf{a_i}$



$$h(o) = \operatorname{argmax}_{a_i \in A} \sum_{t=1}^{T} \left(\log \frac{1-\varepsilon_t}{\varepsilon_t} \right) h_t(o, a_i)$$

$$w(o, a_i) = \sum_{t=1}^{T} \left(\log \frac{1-\varepsilon_t}{\varepsilon_t} \right) h_t(o, a_i)$$


Sensors Chosen by AdaBoost
Accuracy
AdaBoost Iterations
Cluster Size
Training Accuracy
Runtime Accuracy


Average Accuracy
Training Observations per Activity
Training Accuracy
Runtime Accuracy

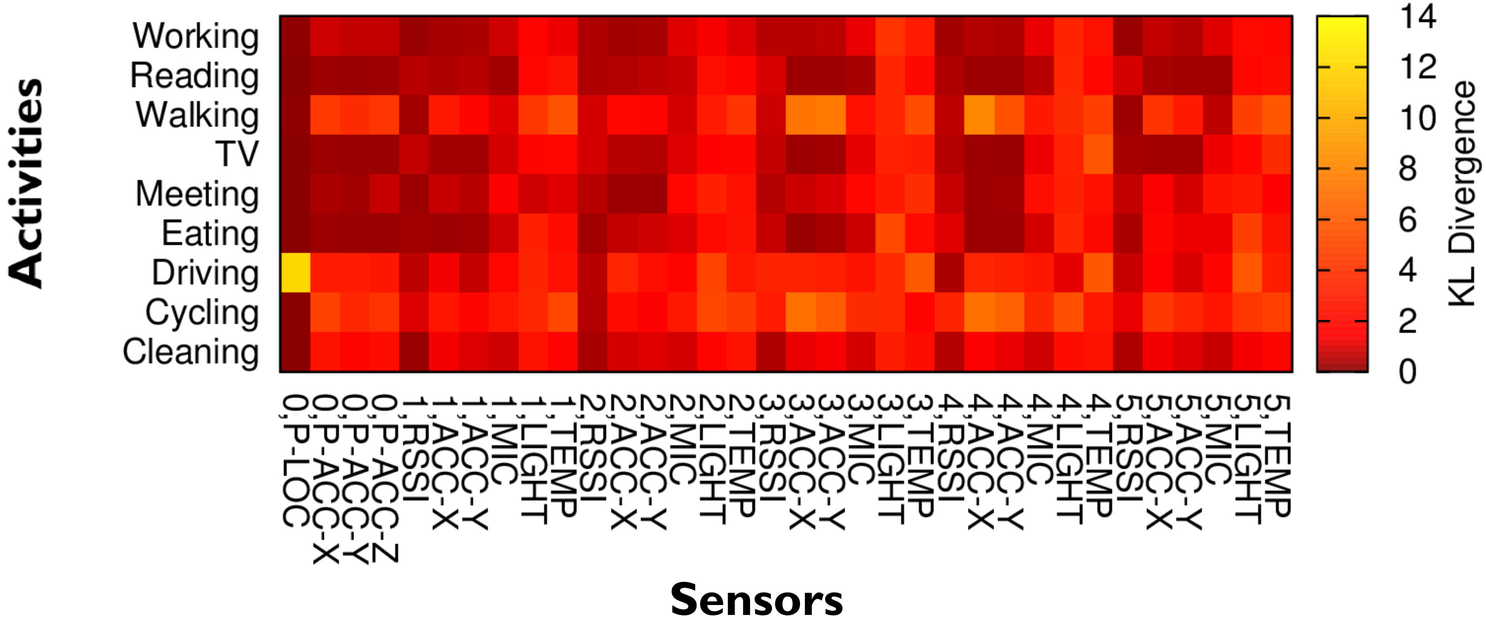
Retraining Detection

- **Body Sensor Network Dynamics** affects accuracy during runtime:
  - Changing physical location
  - User biomechanics
  - Variable sensor orientation
  - Background noise
- Achieve high accuracy with limited initial training data
  - Can also used if existing data is not accurate

- How to detect that retraining is needed without asking for ground truth?
  - Constantly nagging the user for ground truth is annoying
  - Perform with limited initial training data
  - Maintain high accuracy

Retraining Detection

- Measure the discriminative power of each sensor: K-L divergence
  - Quantify the difference between sensor reading distributions



- Retraining detection with K-L divergence:
  - Compare training data to runtime data for each sensor

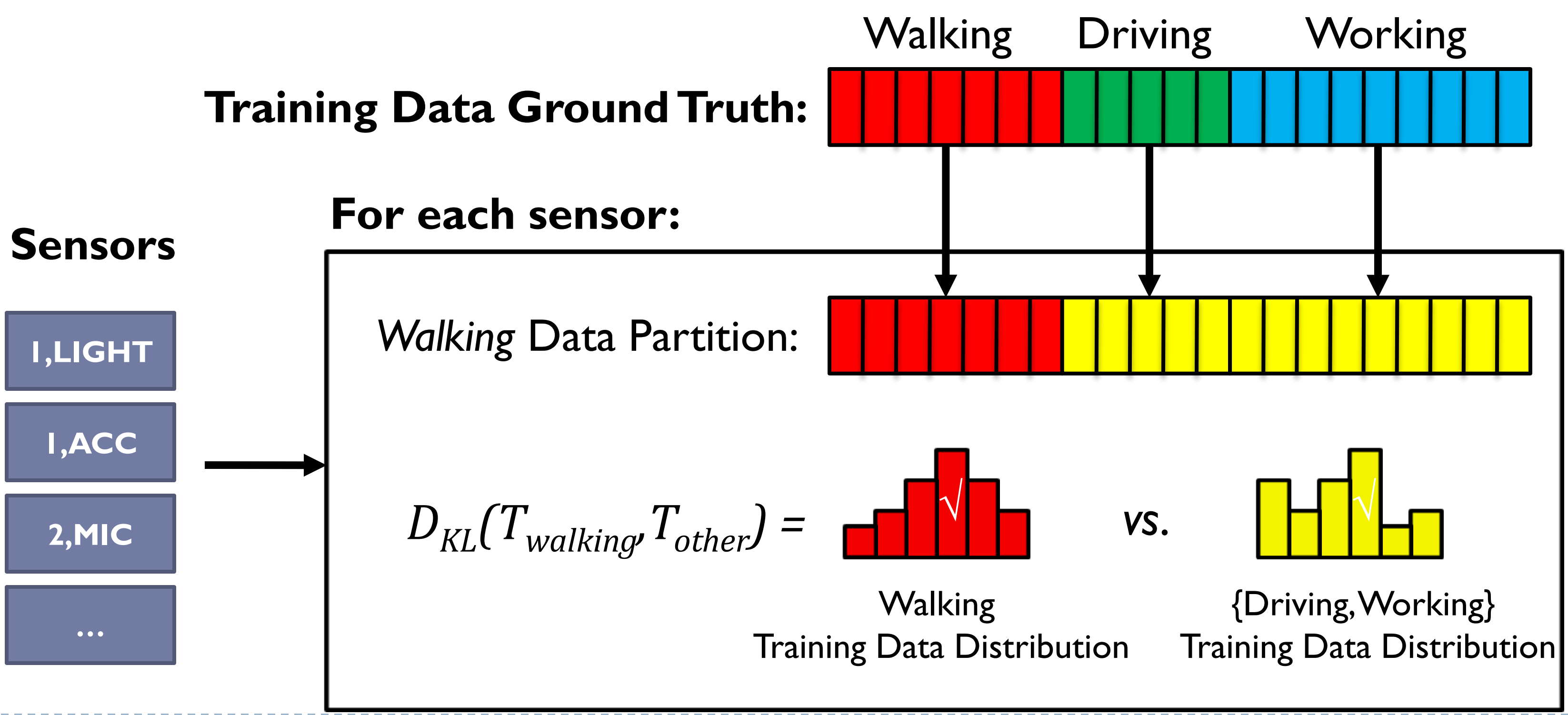
Kullback–Leibler divergence

K-L divergence measures the expected amount of information required to transform samples from a distribution P into a second distribution Q.

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\,\ln\frac{P(i)}{Q(i)}.$$

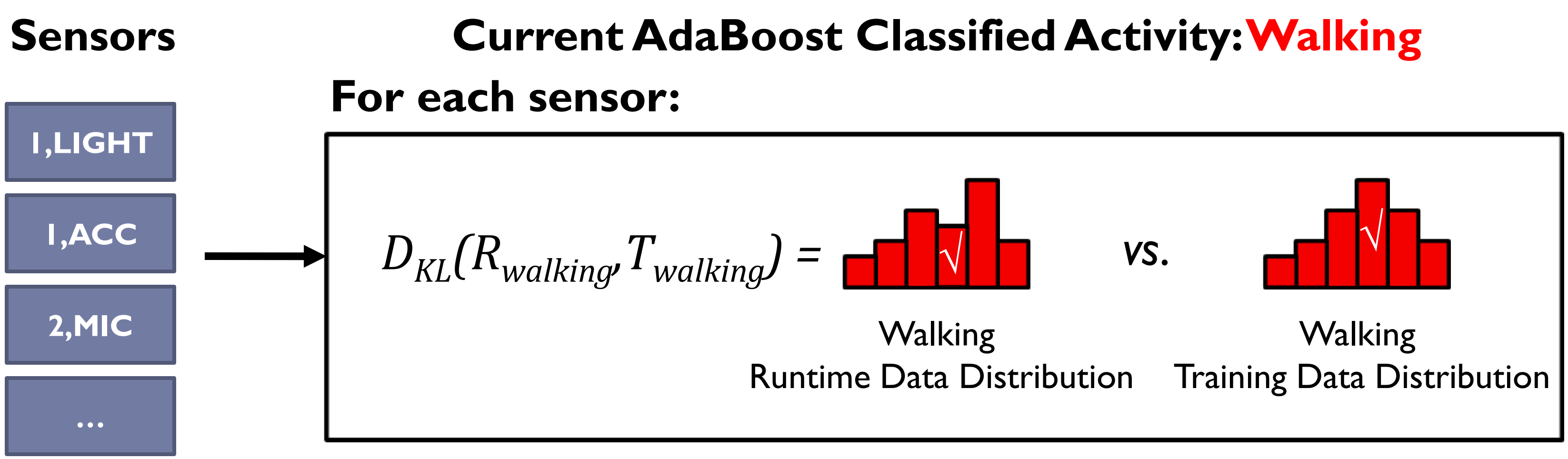
Retraining Detection

- Training
  - Compute "one vs. rest" K-L divergence for each sensor and activity



Retraining Detection

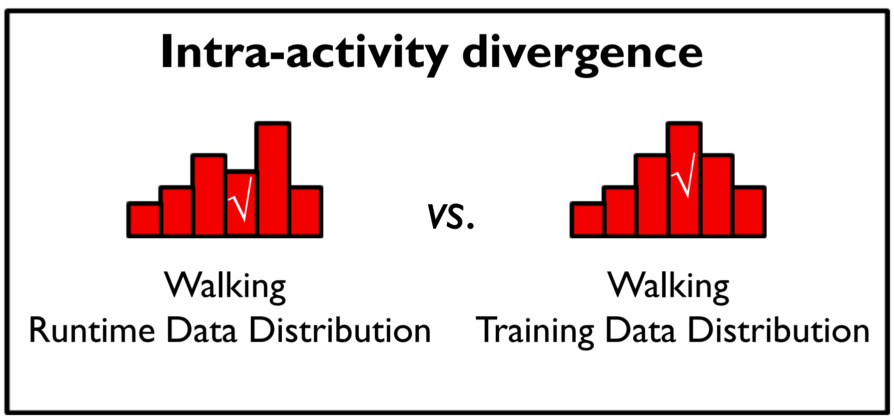
- Runtime
  - At each interval, sensors compare runtime data to training data for current classified activity

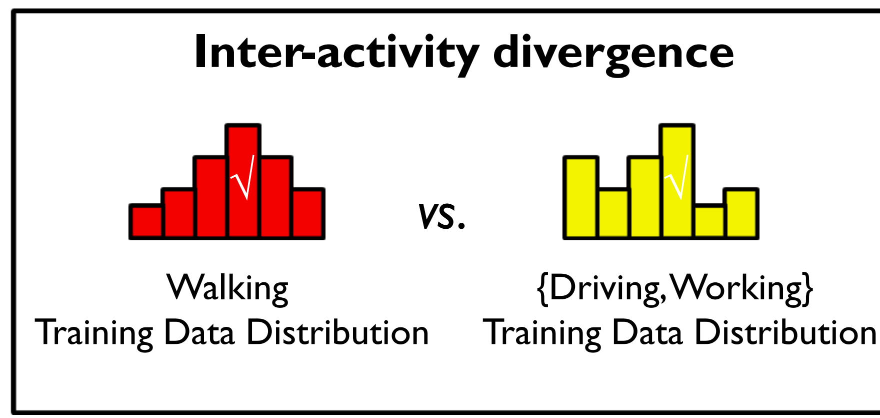


Retraining Detection

- Runtime
  - At each interval, sensors compare runtime data to training data for current classified activity
  - Each individual sensor determines retraining is needed when:

$$D_{KL}(R_{walking}, T_{walking}) \quad > \quad D_{KL}(T_{walking}, T_{other})$$





Retraining Detection

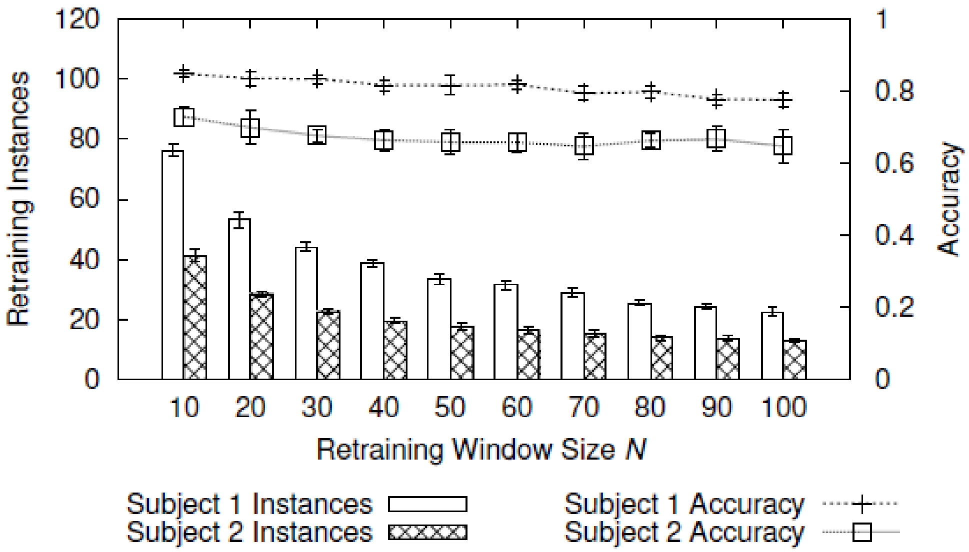
- Runtime
  - At each interval, sensors compare runtime data to training data for current classified activity
  - Each individual sensor determines retraining is needed
  - The ensemble retrains when a **weighted majority** of sensors demand retraining

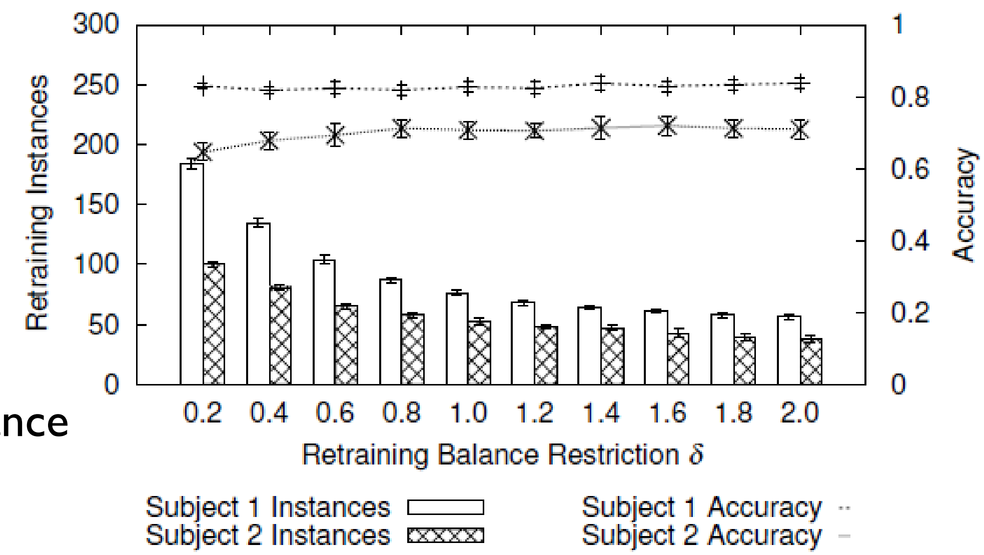
Ground Truth Management

- Retraining: How much new labeled data to collect?
  - Capture changes in body dynamics
  - Too much labeling is intrusive

- Decide to retrain
  - Prompt user to log ground truth for a window of N
  - Use logs the current activity
- Balance number of observations per activity
  - AdaBoost relies on creating weight distribution D for training observations
    - Based on classification difficulty
  - Loose balance hurts classification accuracy
  - Restrictive balance prevents adding new data
  - Balance multiplier
    - Each activity has no more than δ times the average
  - Balance enforcement: random replacement

$$\frac{|O_i| - \frac{1}{|A|}\sum_{\forall a_k \in A} |O_k|}{\frac{1}{|A|}\sum_{\forall a_k \in A} |O_k|} \leq \delta$$





Importance of δ

Further increase does not ensure balance

Sensor Selection

- AdaBoost training can be computationally demanding
  - Train a weak classifier for each sensor at each iteration
  - > 100 iterations to achieve maximum accuracy
- Can we give only the most helpful sensors to AdaBoost?
  - Identify both helpful and redundant sensors
  - Train fewer weak classifiers per AdaBoost iteration
  - Bonus: use even fewer sensors
- Key idea: different weak classifier must have diverse prediction results
  - Less correlation
  - Exclude the redundant sensors

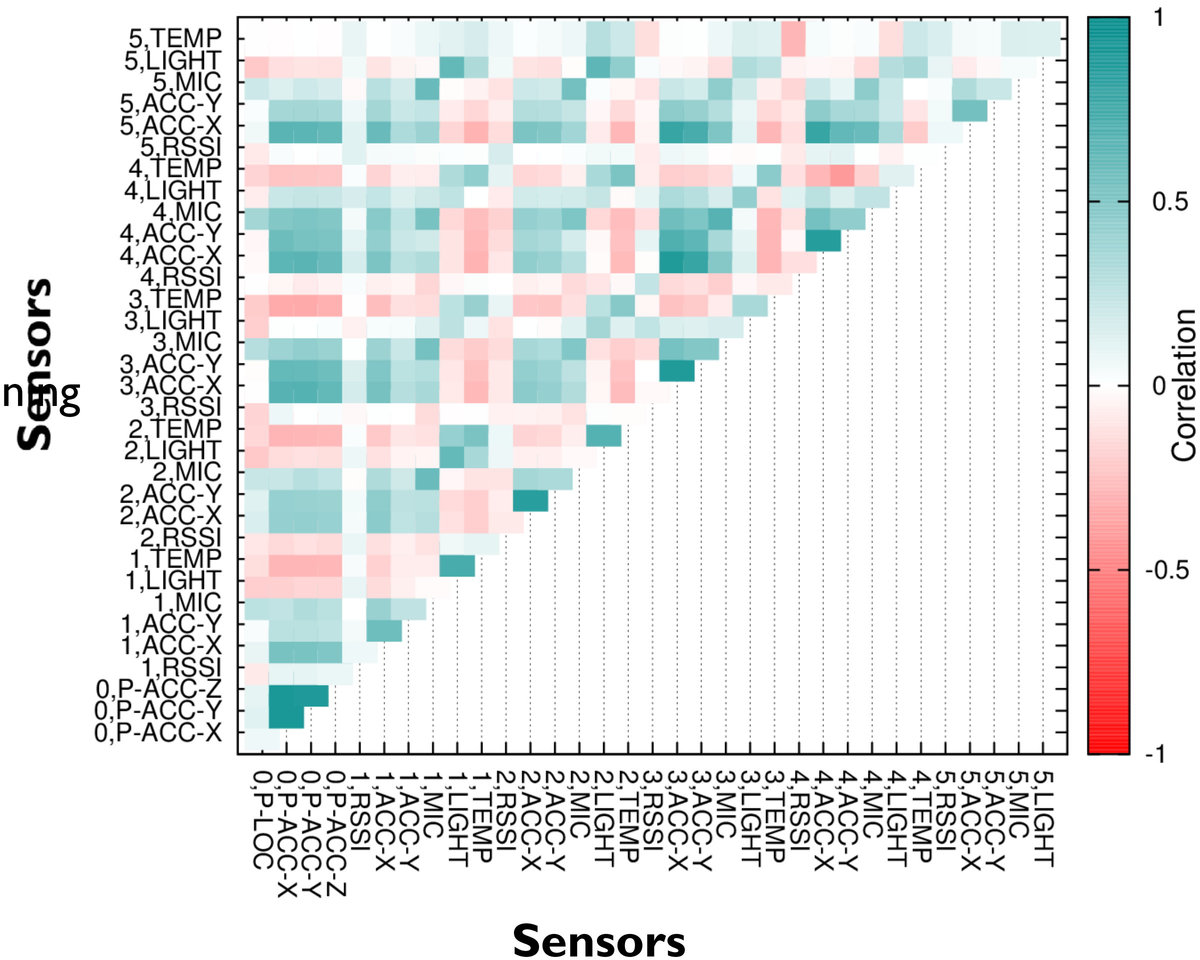
Sensor Selection

Use correlation information between different sensors

Accs, are correlated
Light, temp are correlated

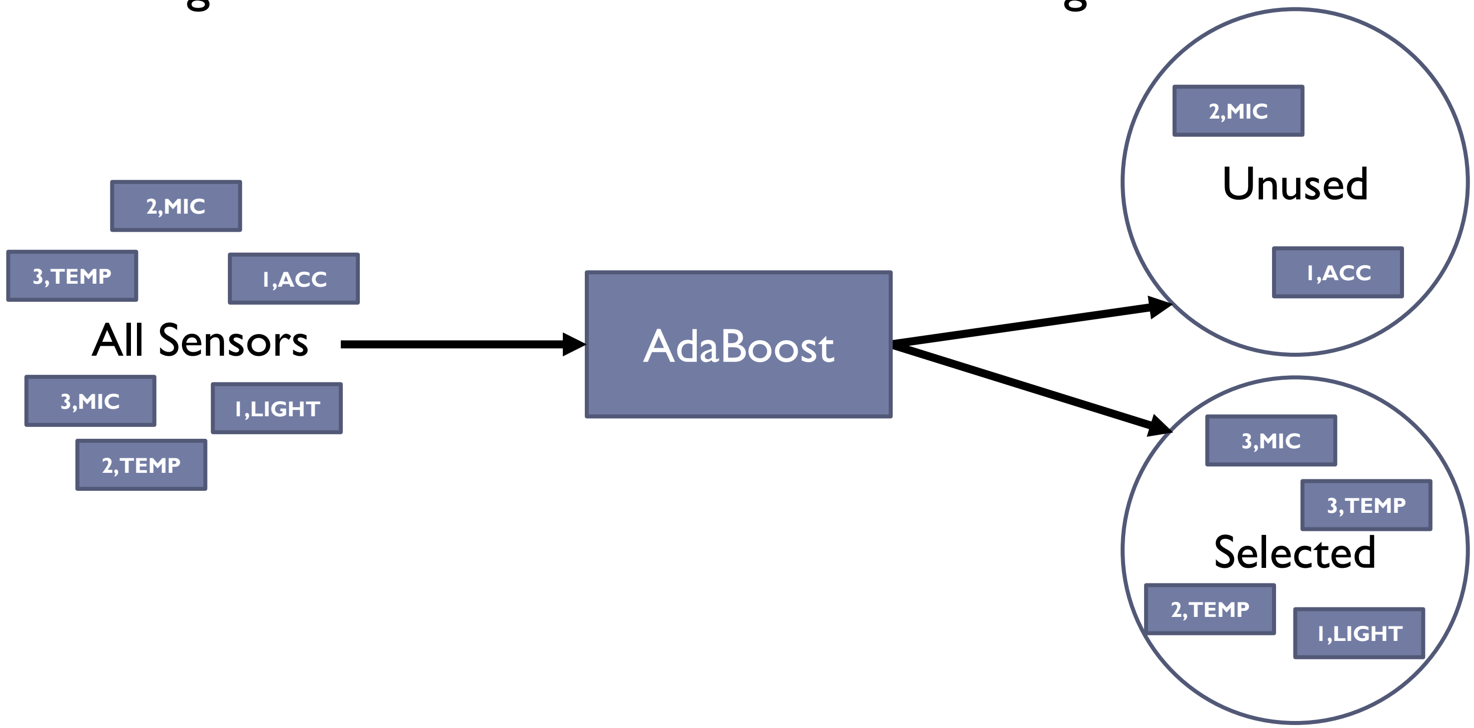
Remove them from AdaBoost training

Raw Data Correlation



Sensor Selection

- Goal: determine the sensors that AdaBoost chooses using correlation
- Find the correlation of each pair of sensors selected by AdaBoost
- Use average correlation as a threshold for choosing sensors



- Sensor selection consists of two components
  - Threshold adjustment
    - Threshold is computed to discriminate the sensors
    - Performed during training
  - Selection
    - Select the set of sensors for retraining

Threshold

- Initialize the threshold during initial training
- Find the correlation between sensors
- Outlier identifies the threshold

Algorithm 2 Raw Correlation Threshold for Sensor Selection

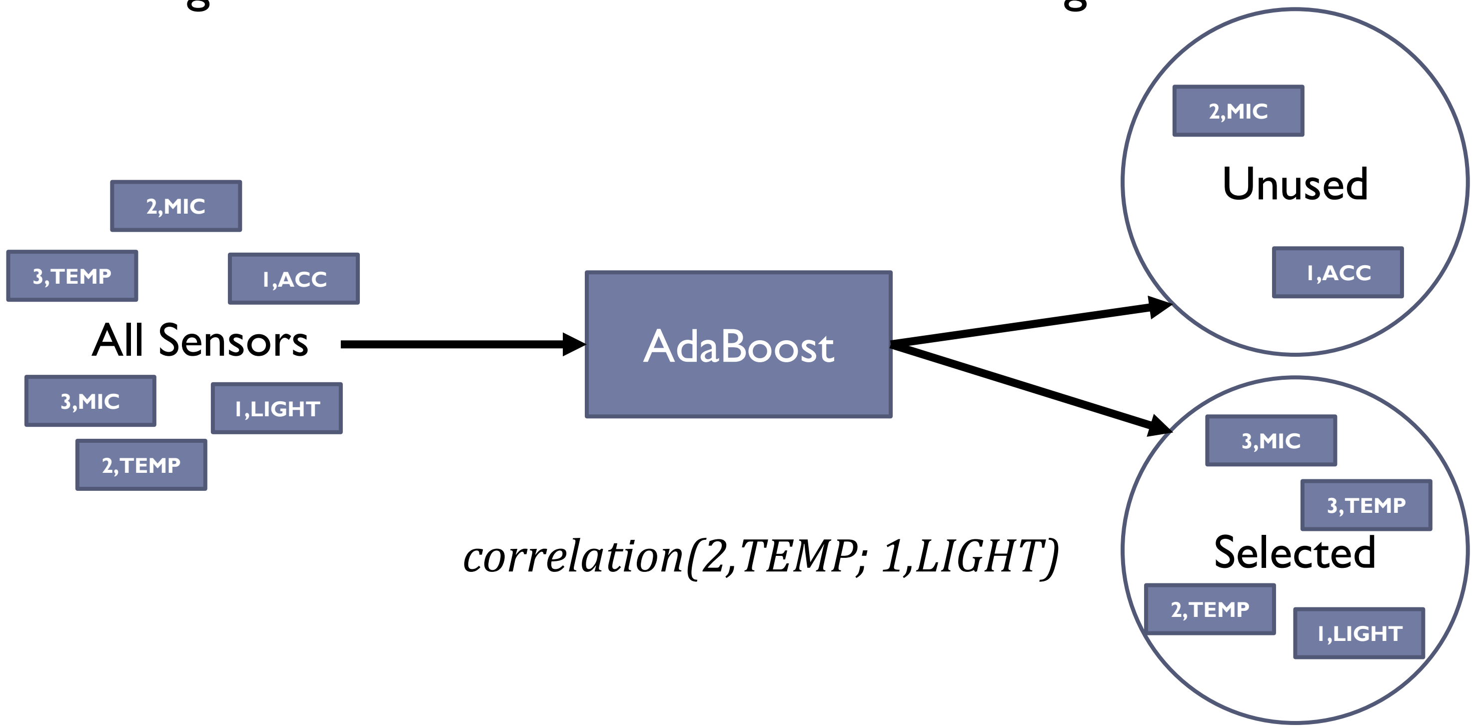
Input: Set of sensors S selected by AdaBoost, training observations for all sensors O, multiplier n

Output: Sensor selection threshold α

```
R = ∅ // set of correlation coefficients
for all combinations of sensors s_i and s_j in S do
    Compute correlation coefficient r = |r_{O_i,O_j}|
    R = R ∪ {r}
end for
// compute threshold as avg + (n * std. dev. ) of R
α = μ_R + nσ_R
```

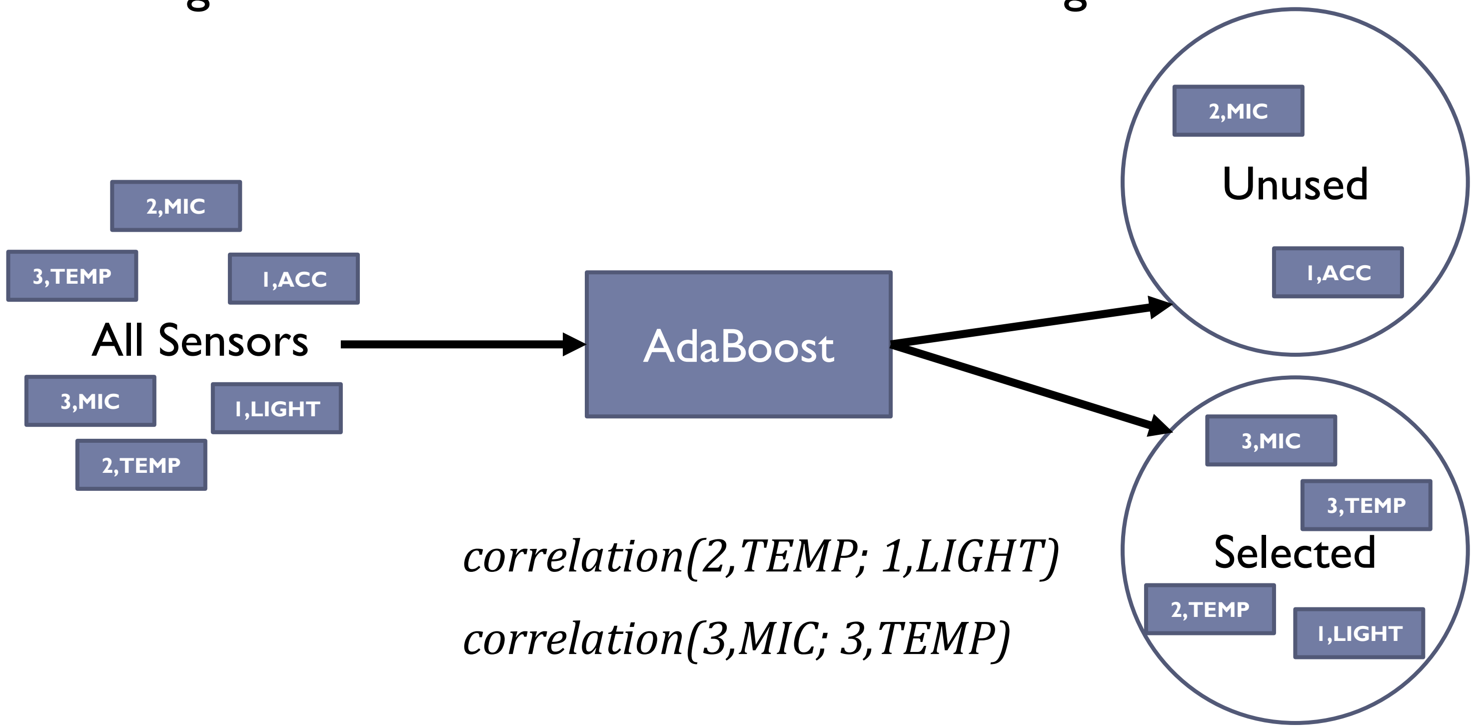
Sensor Selection

- Goal: determine the sensors that AdaBoost chooses using correlation
- Find the correlation of each pair of sensors selected by AdaBoost
- Use average correlation as a threshold for choosing sensors



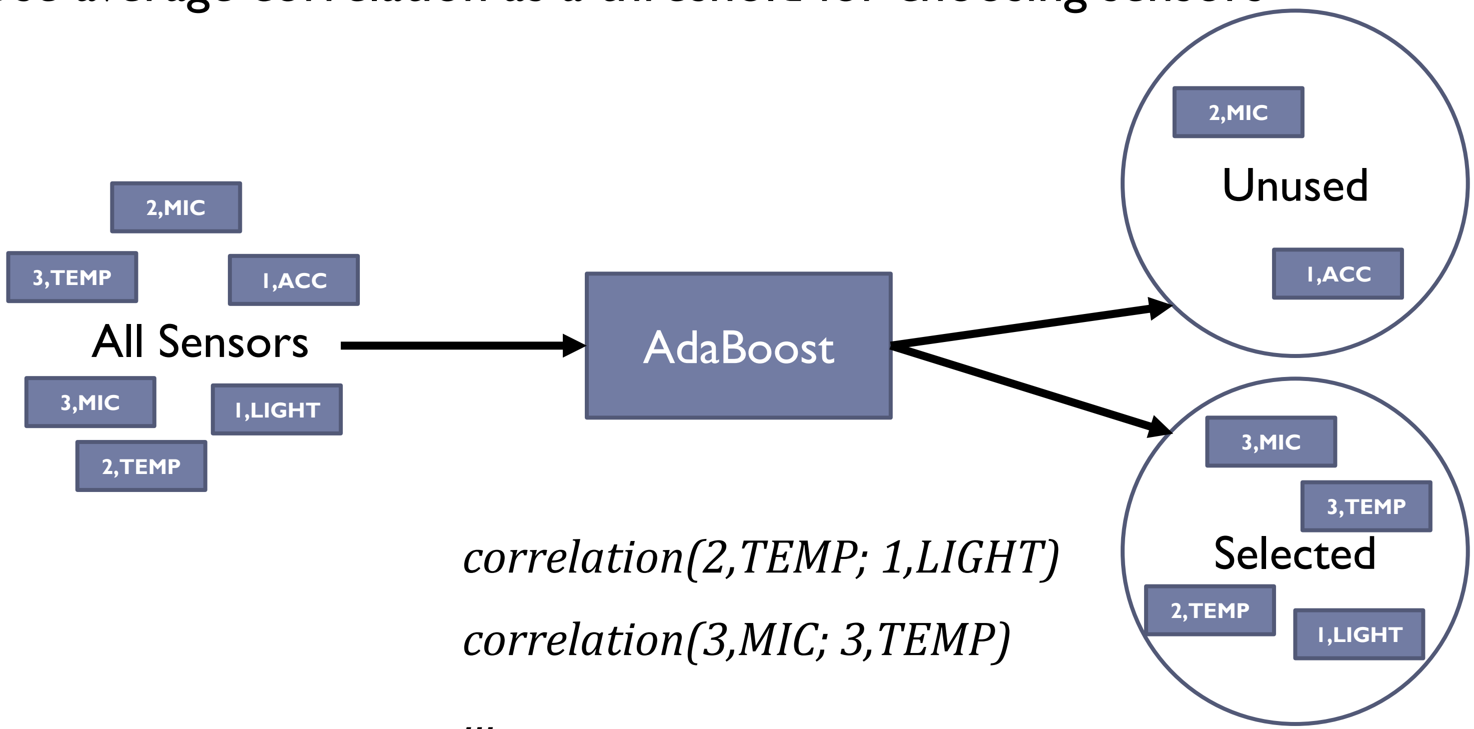
Sensor Selection

- Goal: determine the sensors that AdaBoost chooses using correlation
- Find the correlation of each pair of sensors selected by AdaBoost
- Use average correlation as a threshold for choosing sensors



Sensor Selection

- Goal: determine the sensors that AdaBoost chooses using correlation
- Find the correlation of each pair of sensors selected by AdaBoost
- Use average correlation as a threshold for choosing sensors



correlation(2,TEMP; 1,LIGHT)

correlation(3,MIC; 3,TEMP)

…

Set threshold **α** based on average correlation: $\alpha = \mu_{corr} + \sigma_{corr}$

Selection

- During retraining
  - Choose the set of sensors S* using the threshold α

No two sensors have r>α

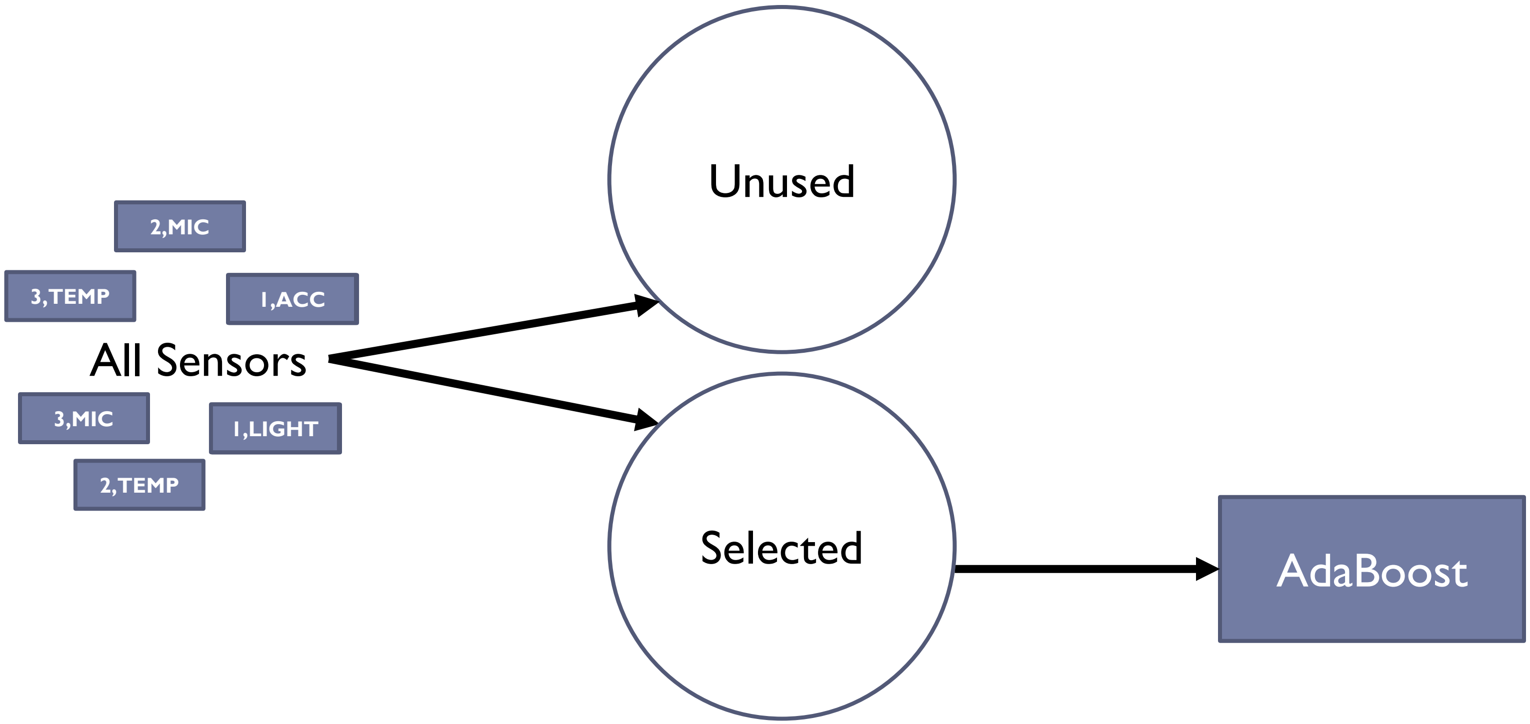
Algorithm 3 Sensor Selection Using Raw Correlation

Input: Set of all sensors S, training observations for all sensors O, threshold α
Output: Selected sensors S^* to give as input to AdaBoost

$S^* = \emptyset$
$E = \emptyset$ *// set of sensors we exclude*
for all combinations of sensors s_i and s_j in S **do**
    Compute correlation coefficient $r = |r_{O_i,O_j}|$
    if $r < \alpha$ **then**
        if $s_i \notin E$ **then** $S^* = S^* \cup \{s_i\}$
        if $s_j \notin E$ **then** $S^* = S^* \cup \{s_j\}$
    else if $r \geq \alpha$ and $\mathrm{acc}(s_i) > \mathrm{acc}(s_j)$ **then**
        // use accuracy to decide which to add to S^*
        if $s_i \notin E$ **then** $S^* = S^* \cup \{s_i\}$
        $E = E \cup \{s_j\}$; $S^* = S^* \setminus \{s_j\}$
    else
        if $s_j \notin E$ **then** $S^* = S^* \cup \{s_j\}$
        $E = E \cup \{s_i\}$; $S^* = S^* \setminus \{s_i\}$
    end if
end for

Sensor Selection

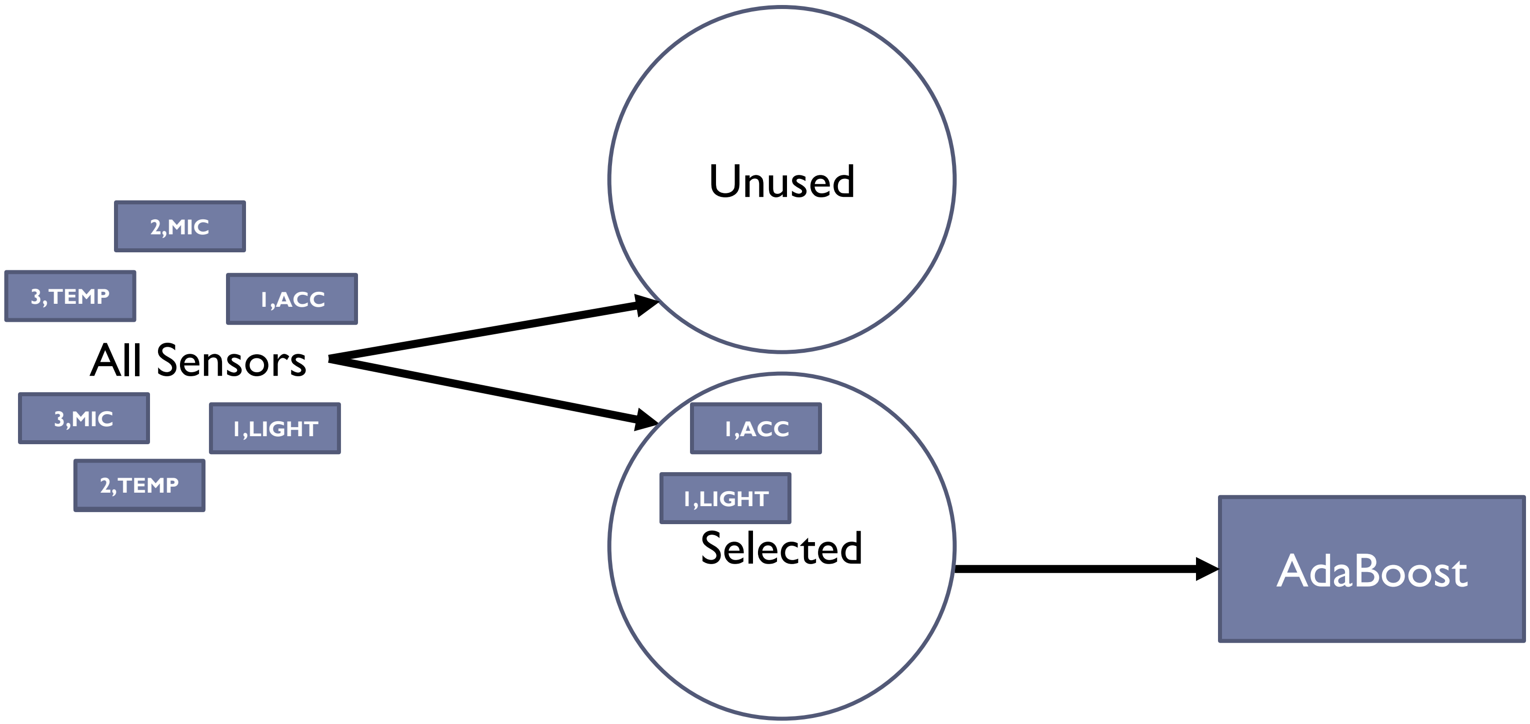
- Choose sensors for input to AdaBoost based on the correlation threshold



$$correlation(1{,}ACC;\ 1{,}LIGHT) \leq \alpha$$

Sensor Selection

- Choose sensors for input to AdaBoost based on the correlation threshold

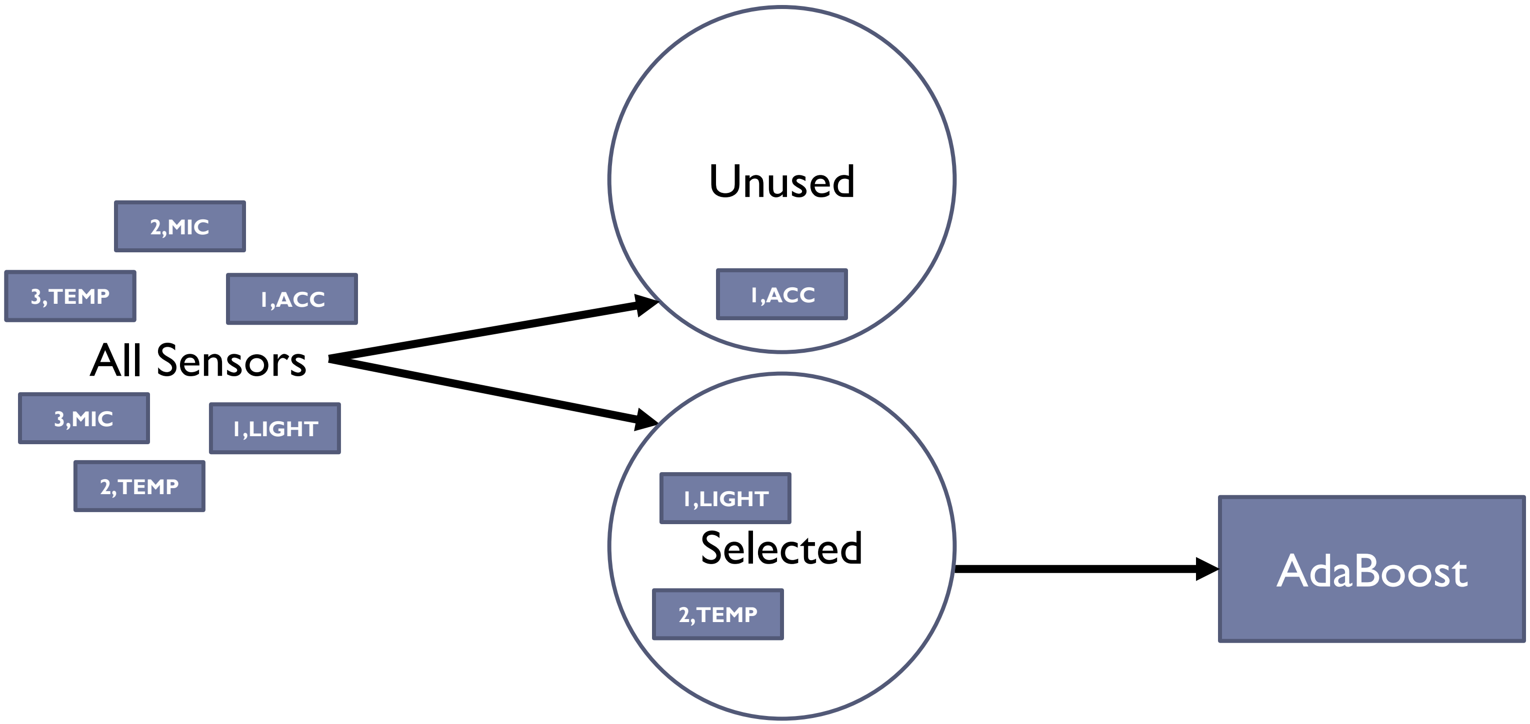


$correlation(2,TEMP;\ 1,ACC) > \alpha$

$acc(2,TEMP) > acc(1,ACC)$

Sensor Selection

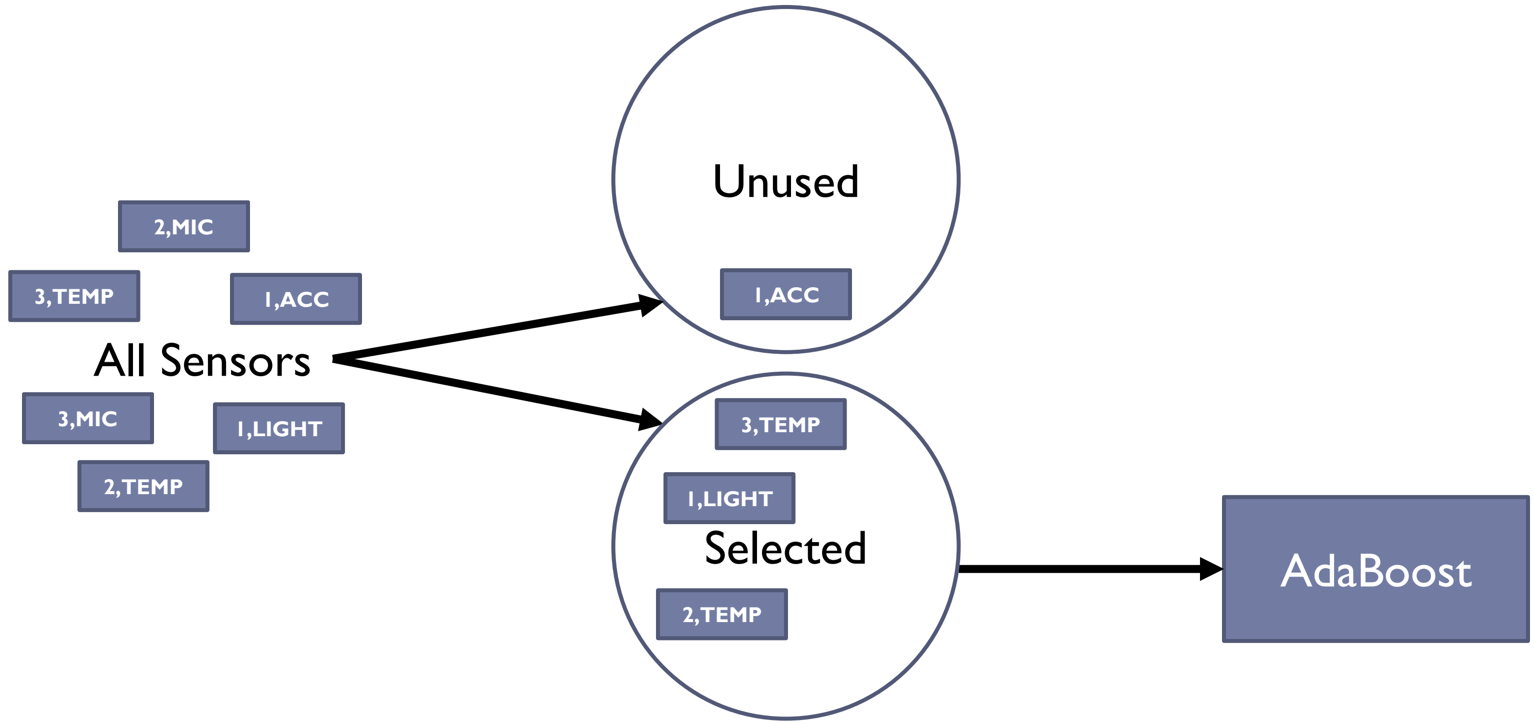
- Choose sensors for input to AdaBoost based on the correlation threshold



$$correlation(1{,}ACC;\ 3{,}TEMP) \leq \alpha$$

Sensor Selection

- Choose sensors for input to AdaBoost based on the correlation threshold

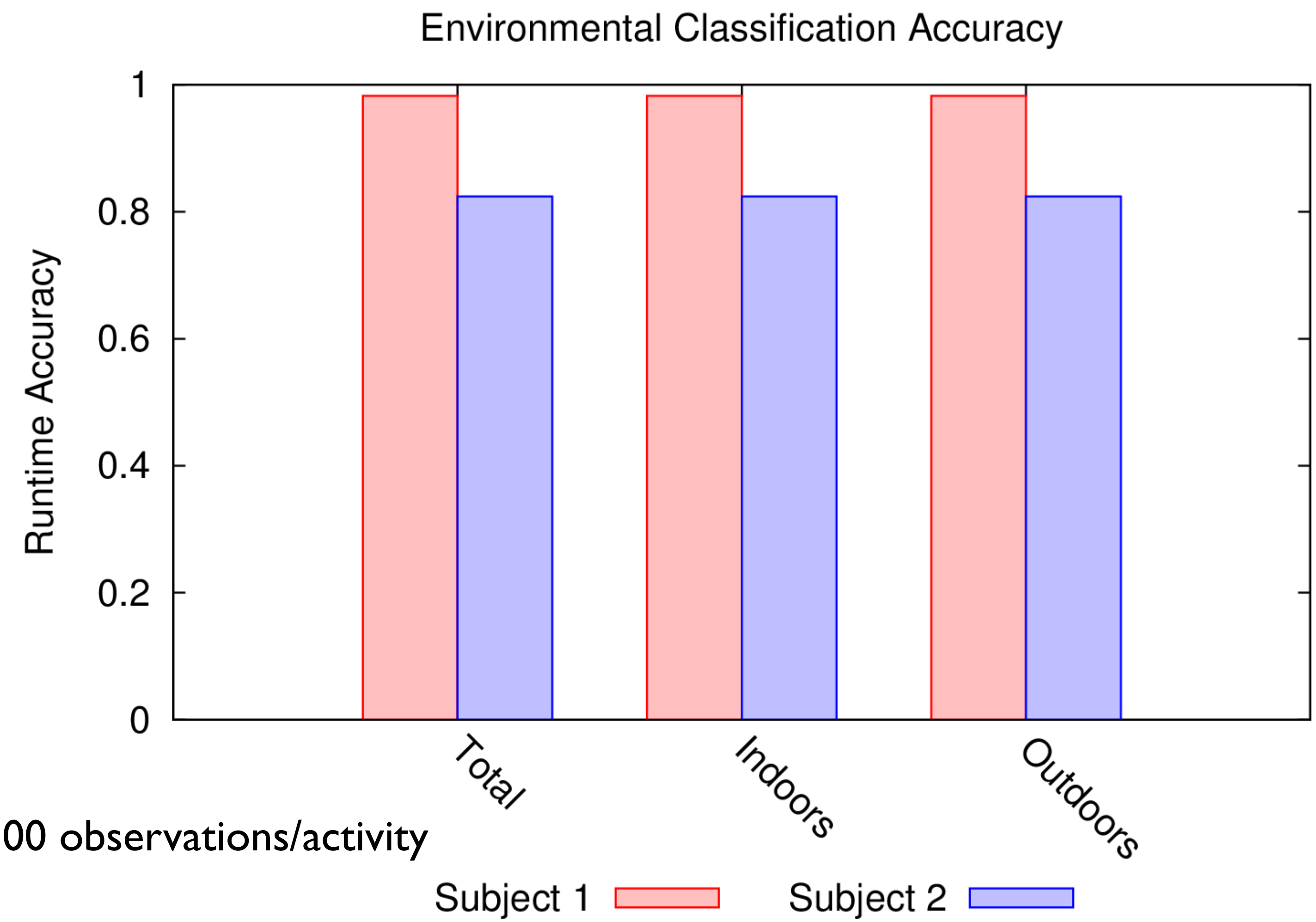


Evaluation Setup

- Classify typical daily activities, postures, and environment
- 2 subjects over 2 weeks
- Classification Categories:

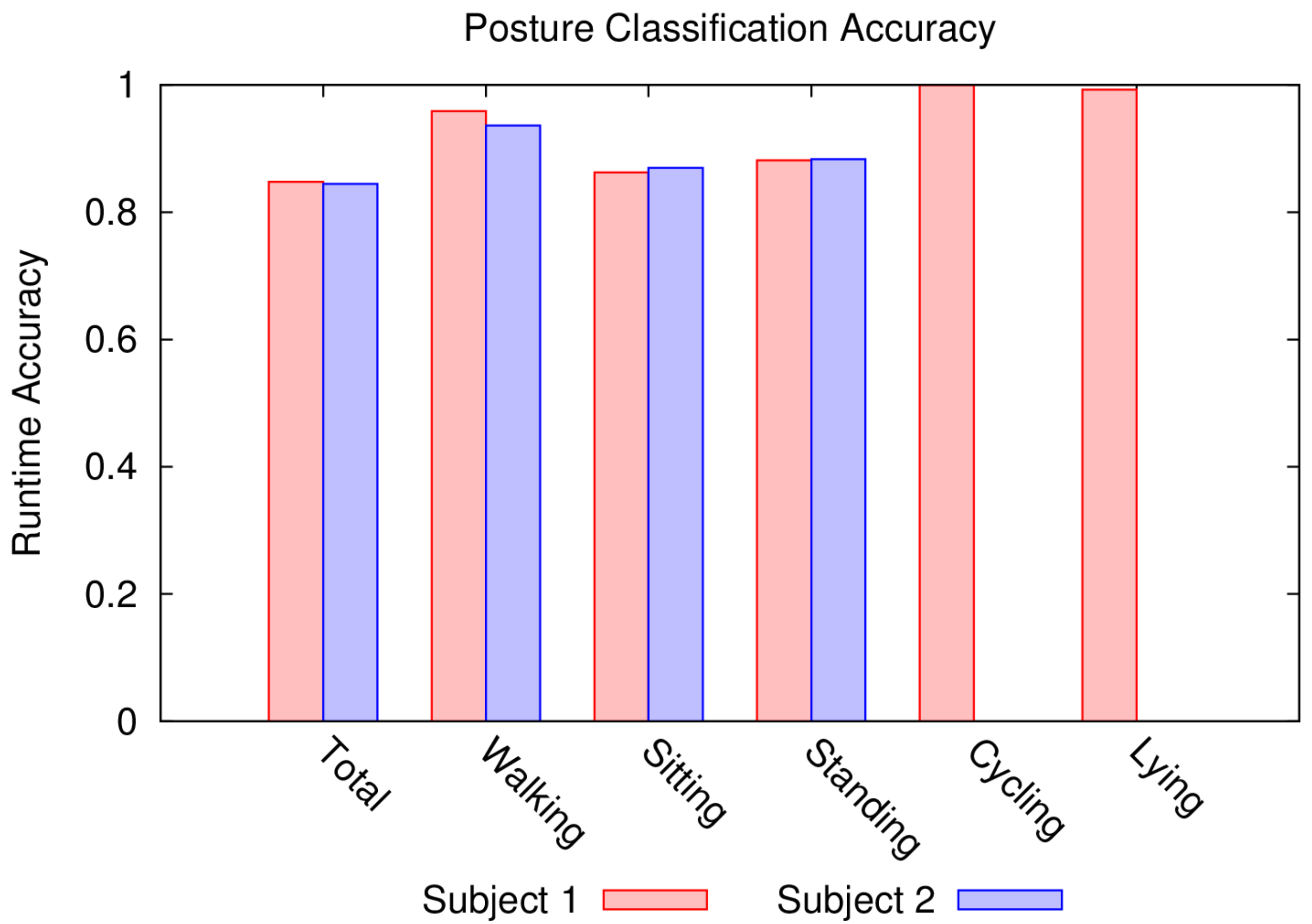
Environment	Indoors, Outdoors
Posture	Cycling, Lying Down, Sitting, Standing, Walking
Activity	Cleaning, Cycling, Driving, Eating, Meeting, Reading, Walking, Watching TV, Working

Classification Performance

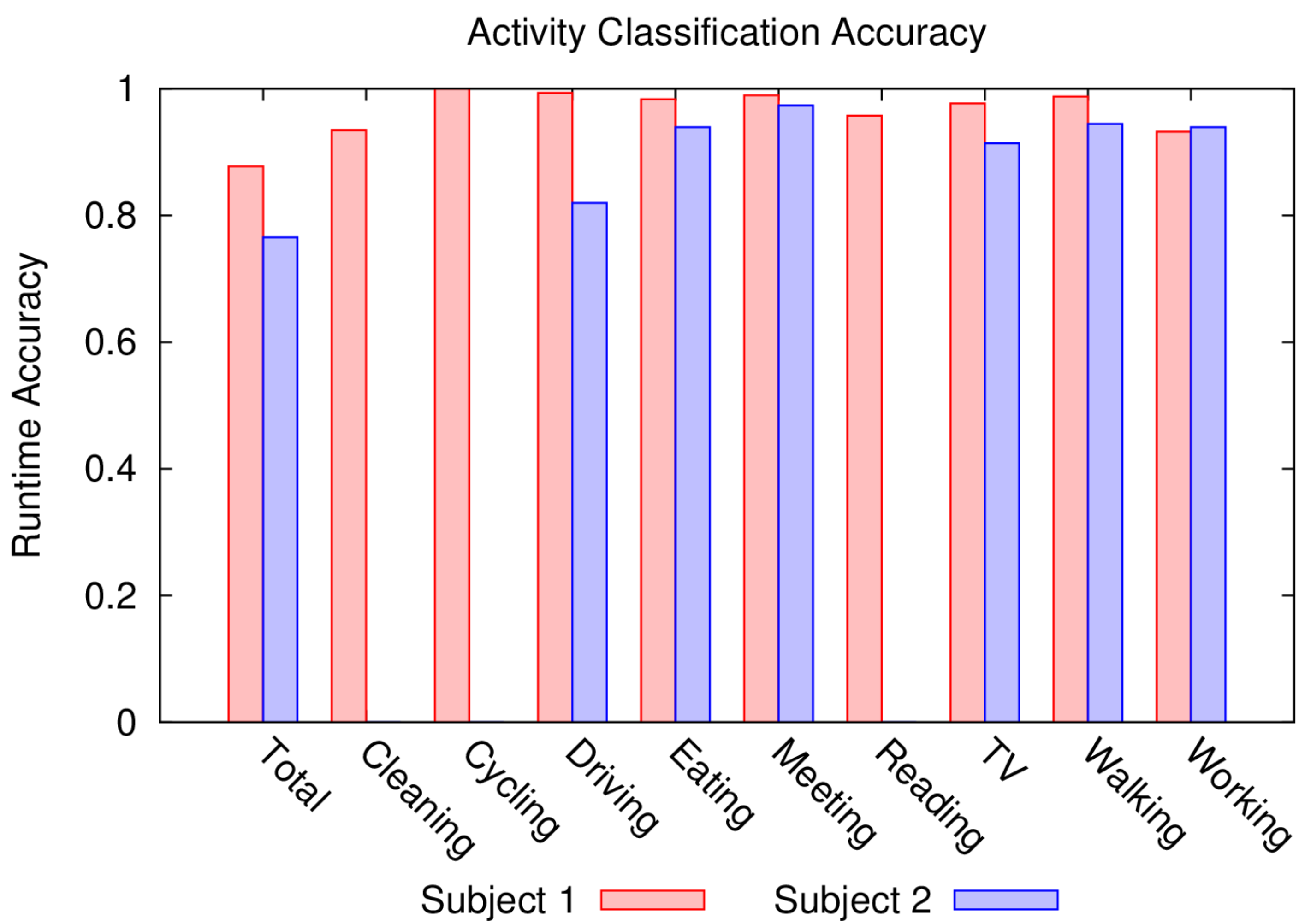


Initial training 100 observations/activity

Classification Performance

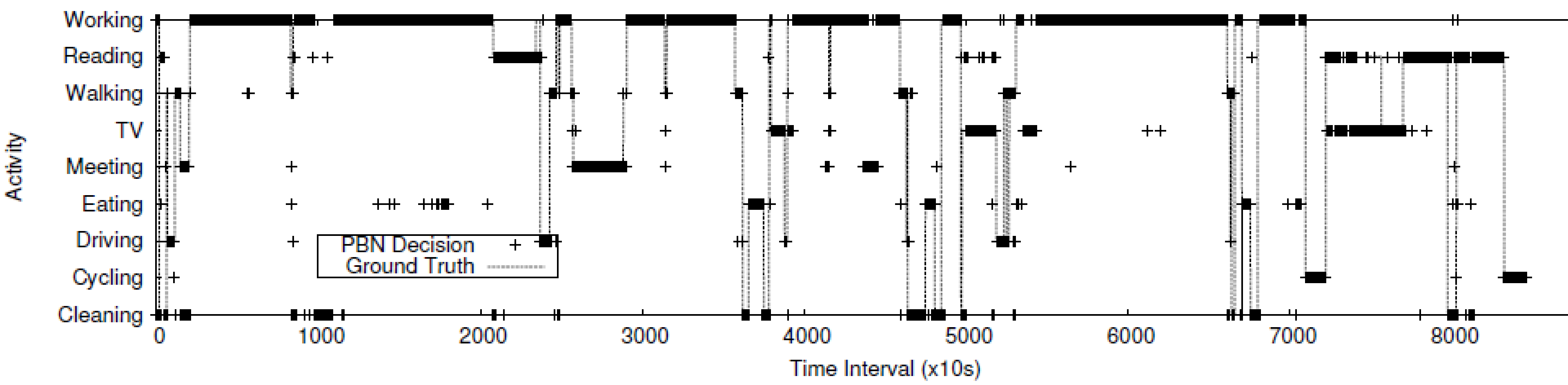


Classification Performance



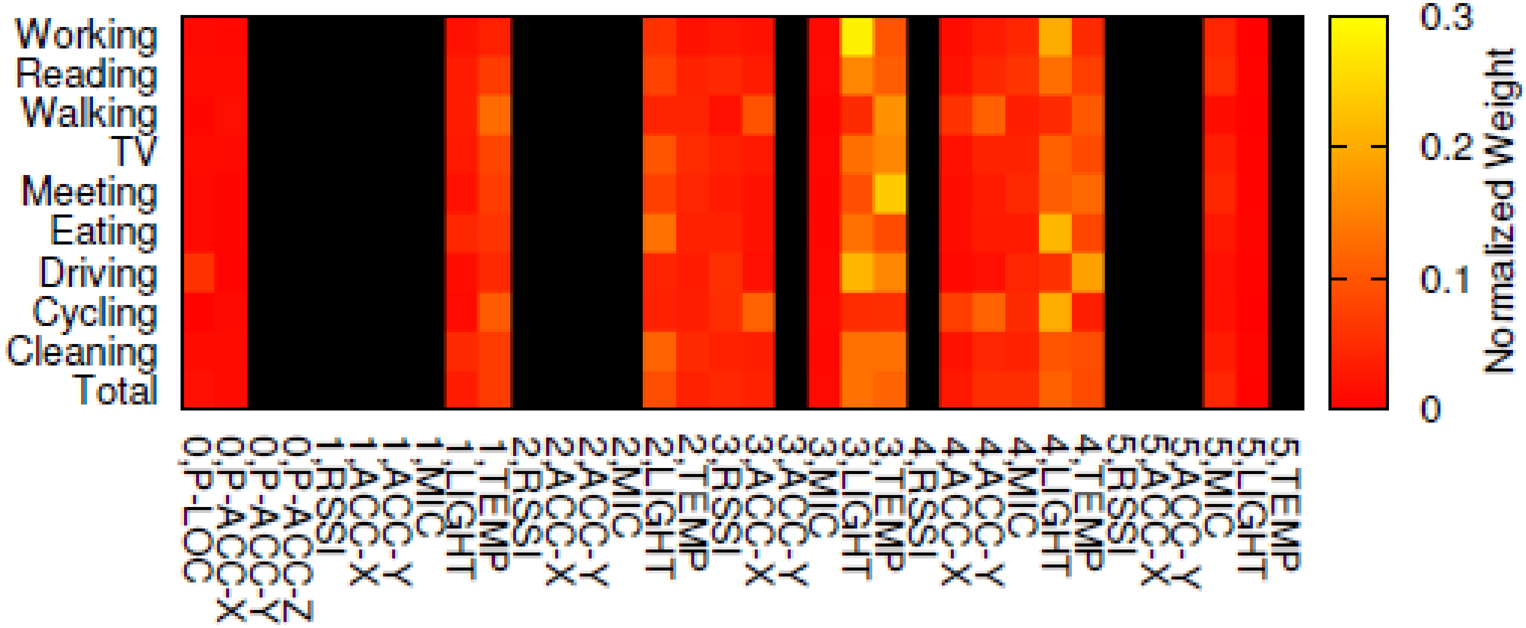
User 1 has accuracy 98%, 85%, 90%

User 2 has accuracy 81%, 82%, 76%

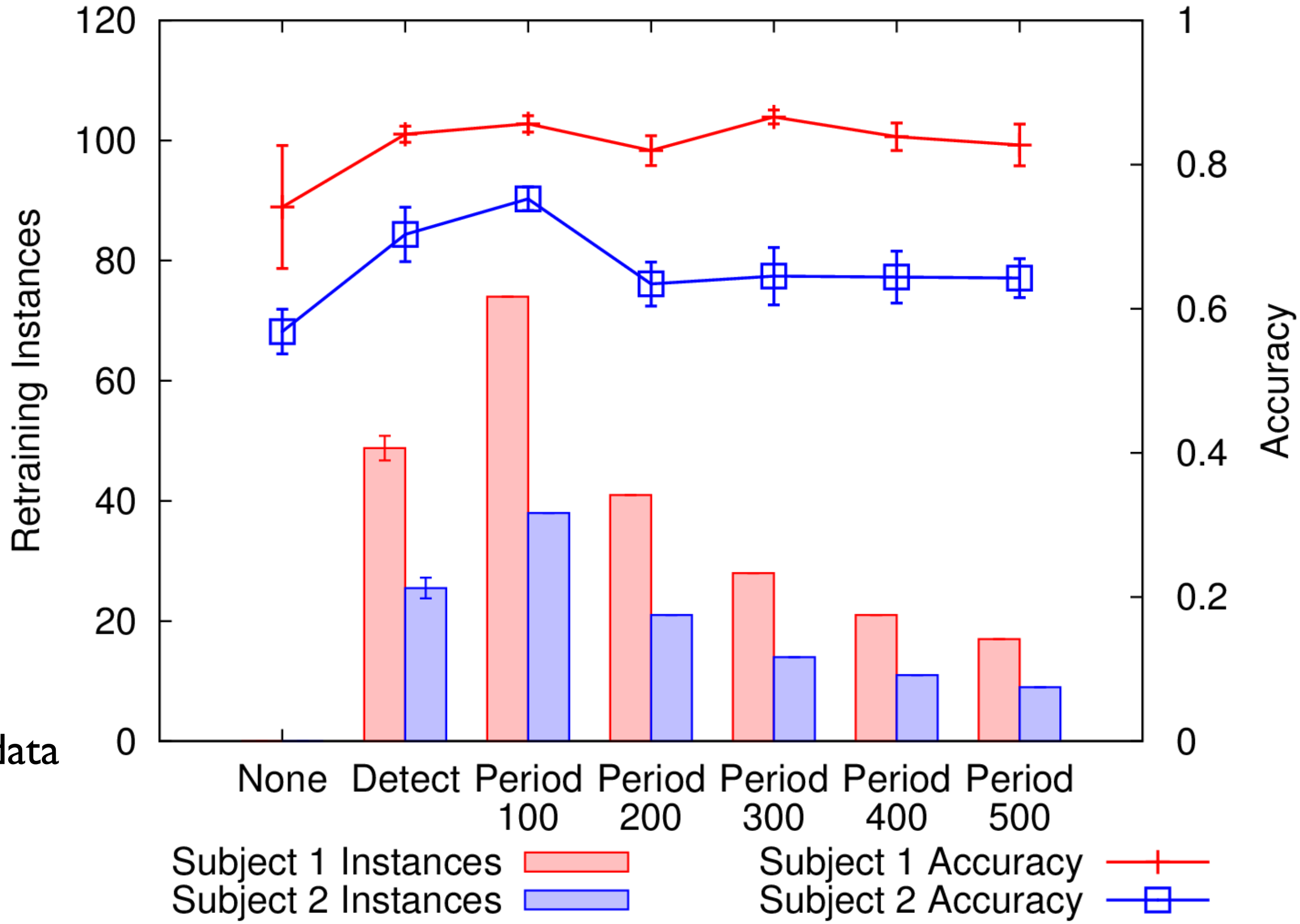


Sensor Weight per activity

16 sensors unused



Retraining Performance



30 new data

Sensor Selection Performance