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:

<table>
<thead>
<tr>
<th>Environment</th>
<th>Indoors, Outdoors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Posture</td>
<td>Cycling, Lying Down, Sitting, Standing, Walking</td>
</tr>
<tr>
<td>Activity</td>
<td>Cleaning, Cycling, Driving, Eating, Meeting, Reading, Walking, Watching TV, Working</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Node ID</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BS/Phone</td>
</tr>
<tr>
<td>1</td>
<td>L. Wrist</td>
</tr>
<tr>
<td>2</td>
<td>R. Wrist</td>
</tr>
<tr>
<td>3</td>
<td>L. Ankle</td>
</tr>
<tr>
<td>4</td>
<td>R. Ankle</td>
</tr>
<tr>
<td>5</td>
<td>Head</td>
</tr>
<tr>
<td>Node</td>
<td>ID</td>
</tr>
<tr>
<td>-------</td>
<td>----</td>
</tr>
<tr>
<td>Phone</td>
<td>0</td>
</tr>
<tr>
<td>IRIS</td>
<td>1</td>
</tr>
<tr>
<td>IRIS</td>
<td>2</td>
</tr>
<tr>
<td>IRIS</td>
<td>3</td>
</tr>
<tr>
<td>IRIS</td>
<td>5</td>
</tr>
</tbody>
</table>

Signal strength
PBN Architecture

Sensor Node

- Local Agg.
- Sensor
- Sample Controller

Base Station Node

- Agg. Data
- Start/Stop
- 802.15.4

Phone

- Sample Controller
- Sensor
- Local Agg.

- GUI

- TinyOS Comm. Stack

- Labeled Data
- Activity Decision, Request Ground Truth
- Agg. Data

- Retraining Detection
- Ground Truth Management
- Activity Classification
- Sensor Selection

- Activity Prob, Agg. Data
- Input Sensors

- USB

- PBN Architecture

- Sensor Node

- Base Station Node

- Phone

- Sample Controller

- Sensor

- Local Agg.

- GUI

- TinyOS Comm. Stack

- Labeled Data

- Activity Decision, Request Ground Truth

- Agg. Data

- Retraining Detection

- Ground Truth Management

- Activity Classification

- Sensor Selection

- Activity Prob, Agg. Data

- Input Sensors

- USB
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 $H$

1: Initialize observation weights $D_1$ to $1/n$ for all obs.
2: $\textbf{for } t = 1 \textbf{ to } T \textbf{ do}$
3: $\quad \textbf{for } \text{sensor } s_j \in S \textbf{ do}$
4: $\quad \quad$ Train weak classifier $h_{t,j}$ using obs. $O_j$, weights $D_t$
5: $\quad \quad$ Get weighted error $\varepsilon_{t,j}$ for $h_{t,j}$ using labels $[8]$ 
6: $\quad \textbf{end for}$
7: $\quad$ Add the $h_{t,j}$ with least error $\varepsilon_t$ to $H$ by choosing $h_{t,j}$ with least error $\varepsilon_t$
8: $\quad$ Set $D_{t+1}$ using $D_t$, misclassifications made by $h_t$ $[8]$
9: $\textbf{end for}$
Final outcome of AdaBoost

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

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

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

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

- **Training**
  - Compute “one vs. rest” K-L divergence for each sensor and activity

![Diagram showing training data ground truth and data partitions for walking, driving, and working activities.](image-url)

For each sensor:

\[ D_{KL}(T_{\text{walking}}, T_{\text{other}}) = \sqrt{\text{Walking Training Data Distribution}} \quad \text{vs.} \quad \sqrt{\text{\{Driving, Working\} Training Data Distribution}} \]
Retraining Detection

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

For each sensor:

\[
D_{KL}(R_{\text{walking}}, T_{\text{walking}}) = \sqrt{\text{Walking Runtime Data Distribution}} \quad \text{vs.} \quad \sqrt{\text{Walking Training Data Distribution}}
\]
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:

\[
D_{KL}(R_{\text{walking}}, T_{\text{walking}}) > D_{KL}(T_{\text{walking}}, T_{\text{other}})
\]

- **Intra-activity divergence**
  - Walking Runtime Data Distribution vs. Walking Training Data Distribution

- **Inter-activity divergence**
  - Walking Training Data Distribution vs. \{Driving, Working\} Training Data Distribution
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 \( \delta \) 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 $\delta$
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
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 $\alpha$

1. $R = \emptyset$ // set of correlation coefficients
2. for all combinations of sensors $s_i$ and $s_j$ in $S$ do
3.  Compute correlation coefficient $r = |r_{o_i,o_j}|$
4.  $R = R \cup \{r\}$
5. end for
6. // compute threshold as avg + (n * std. dev.) of $R$
7. $\alpha = \mu_R + n\sigma_R$
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

![Diagram showing sensor selection process]

\[ \text{correlation}(2, \text{TEMP}; 1, \text{LIGHT}) \]
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

\[ \text{correlation}(2, \text{TEMP}; 1, \text{LIGHT}) \]
\[ \text{correlation}(3, \text{MIC}; 3, \text{TEMP}) \]
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 $\alpha$ based on average correlation: $\alpha = \mu_{corr} + \sigma_{corr}$
Selection

- During retraining
  - Choose the set of sensors $S^*$ using the threshold $\alpha$

No two sensors have $r > \alpha$

Algorithm 3: Sensor Selection Using Raw Correlation

Input: Set of all sensors $S$, training observations for all sensors $O$, threshold $\alpha$

Output: Selected sensors $S^*$ to give as input to AdaBoost

1: $S^* = \emptyset$
2: $E = \emptyset$ // set of sensors we exclude
3: for all combinations of sensors $s_i$ and $s_j$ in $S$ do
4:   Compute correlation coefficient $r = |r_{O_i,O_j}|$
5:   if $r < \alpha$ then
6:     if $s_i \notin E$ then $S^* = S^* \cup \{s_i\}$
7:     if $s_j \notin E$ then $S^* = S^* \cup \{s_j\}$
8:     else if $r \geq \alpha$ and $\text{acc}(s_i) > \text{acc}(s_j)$ then
9:       // use accuracy to decide which to add to $S^*$
10:      if $s_i \notin E$ then $S^* = S^* \cup \{s_i\}$
11:      $E = E \cup \{s_j\}; S^* = S^* \\{s_j\}$
12:     else
13:      if $s_j \notin E$ then $S^* = S^* \cup \{s_j\}$
14:      $E = E \cup \{s_i\}; S^* = S^* \\{s_i\}$
15:   end if
16: end for
Sensor Selection

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

\[ \text{correlation}(1,\text{ACC}; 1,\text{LIGHT}) \leq \alpha \]
Sensor Selection

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

\[
\text{correlation}(2, \text{TEMP}; 1, \text{ACC}) > \alpha \\
\text{acc}(2, \text{TEMP}) > \text{acc}(1, \text{ACC})
\]
Sensor Selection

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

\[ \text{correlation}(1, \text{ACC}; 3, \text{TEMP}) \leq \alpha \]
Sensor Selection

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

All Sensors

- 2,MIC
- 3,TMP
- 3,MIC
- 2,TMP

Unused

- 1,ACC

Selected

- 1,LIGHT

AdaBoost

- 3,TMP
- 2,TMP

- 1,LIGHT
Evaluation Setup

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

<table>
<thead>
<tr>
<th>Environment</th>
<th>Indoors, Outdoors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Posture</td>
<td>Cycling, Lying Down, Sitting, Standing, Walking</td>
</tr>
<tr>
<td>Activity</td>
<td>Cleaning, Cycling, Driving, Eating, Meeting, Reading, Walking, Watching TV, Working</td>
</tr>
</tbody>
</table>
Classification Performance

Environmental Classification Accuracy

Run Time Accuracy

Initial training 100 observations/activity
Classification Performance

Posture Classification Accuracy

Runtime Accuracy

Total  Walking  Sitting  Standing  Cycling  Lying
Subject 1  Subject 2

0  0.2  0.4  0.6  0.8  1
Classification Performance

Activity Classification Accuracy

Runtime Accuracy

Subject 1  Subject 2
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

Retraining Instances

Accuracy

Subject 1 Instances

Subject 2 Instances

Subject 1 Accuracy

Subject 2 Accuracy
Sensor Selection Performance

The bar chart illustrates the percentage of sensors excluded for different methods: AdaBoost Only, SS Only, and AdaBoost+SS. The chart compares the performance across two subjects, marked as Subject 1 in the legend. The data suggests that AdaBoost+SS generally excludes a higher percentage of sensors compared to the other methods.