EFFICIENT MACHINE LEARNING ALGORITHMS FOR AUTOMATIC RECONFIGURATION OF MOBILE HEALTH MONITORING SYSTEMS

By

RAMYAR SAEEDI

A dissertation submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

JULY 2018

© Copyright by RAMYAR SAEEDI, 2018
All Rights Reserved
To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of RAMYAR SAEEDI find it satisfactory and recommend that it be accepted.

__________________________________________________________________
Assefaw H. Gebremedhin, Ph.D., Chair

__________________________________________________________________
José G. Delgado-Frias, Ph.D.

__________________________________________________________________
Sandip Roy, Ph.D.
I would like to express my deepest gratitude to my supervisor Professor Assefaw Gebremedhin for his great support, valuable advice and constant encouragement during my PhD program. I gratefully acknowledge my PhD committee members, Professor Jose Delgado and Professor Sandip Roy for their comments and suggestions that have greatly improved the thesis. I also acknowledge Professor Hassan Ghasemzadeh for his valuable advice. I would like to thank my research collaborators, Keyvan Sasani and Skyler Norgaard, in SCADS lab for their expertise and help through my PhD.
EFFICIENT MACHINE LEARNING ALGORITHMS FOR
AUTOMATIC RECONFIGURATION OF MOBILE
HEALTH MONITORING SYSTEMS

Abstract

by Ramyar Saeedi, Ph.D.
Washington State University
July 2018

Chair: Assefaw H. Gebremedhin

Mobile health monitoring plays a central role in a variety of healthcare applications. Due to the sensitive nature of healthcare applications, these systems need to process these personal and physiological informations highly accurately. However, as mobile devices are employed in real-world scenarios, the accuracy of a machine learning model drops whenever a change in the setting of the system occurs. Therefore, machine learning algorithms that specifically address challenges associated with dynamic real-world situations are needed. At a high level, the variations can be grouped in three classes: configuration change (e.g. sensor sensitivity), context change (e.g. new user), and user need change (e.g. new sensor deployment). To cope with the effects
of the above variations, we develop efficient machine learning algorithms for designing robust and reconfigurable embedded software for these systems. We envision a high level framework for health monitoring systems and we take several inter-related directions for realizing the framework. First, we develop several algorithms to adapt the current knowledge for the setting at hand autonomously. These algorithms find relation between the setting at hand and training data, and generate context-specific labeled data. Secondly, for the case where related knowledge is insufficient to reconfigure the system, we propose a novel architecture called Co-MEAL designed for interactive reconfiguration of the system. The key idea is to keep the systems uncertainty below a pre-specified threshold while minimizing the overall cost of data annotation. This framework is designed to manage a set of heterogeneous experts and provide collaboration between the experts. Thirdly, we proposed a novel active deep architecture, 3DConvLSTM, to generate generalizable representations from raw-data. The architecture learns representations from related data, and capture the time dependency between consecutive instances in the active learning phase. Lastly, to control the way we perform reconfiguration, we propose an efficient management framework to be used in dynamically changing network of wearables. We introduce a novel graph model to capture the importance of features and their correlations along with their associated cost. We propose a greedy algorithm to solve the minimum-cost feature selection problem to classify the uncertainty (signal heterogeneity) situations.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>x</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Mobile Health Monitoring</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Design Requirements</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Thesis Contribution</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Publications</td>
<td>9</td>
</tr>
<tr>
<td>1.5 Organization of The Thesis</td>
<td>11</td>
</tr>
<tr>
<td>2. Preliminaries</td>
<td>12</td>
</tr>
<tr>
<td>2.1 Signal Processing</td>
<td>12</td>
</tr>
<tr>
<td>2.2 Transfer Learning</td>
<td>15</td>
</tr>
<tr>
<td>2.3 Active Learning</td>
<td>17</td>
</tr>
<tr>
<td>2.4 Deep Learning</td>
<td>20</td>
</tr>
<tr>
<td>3. Existing Approaches for Dealing with Data Variation</td>
<td>22</td>
</tr>
<tr>
<td>3.1 Reconfigurable Learning Approaches</td>
<td>22</td>
</tr>
<tr>
<td>3.2 Robust Supervised Learning Approaches</td>
<td>24</td>
</tr>
<tr>
<td>3.3 Semi/Unsupervised Learning Approaches</td>
<td>27</td>
</tr>
<tr>
<td>4. Motivation for Reconfiguration</td>
<td>29</td>
</tr>
<tr>
<td>4.1 Scenarios Driving Reconfiguration</td>
<td>29</td>
</tr>
</tbody>
</table>
4.2 Description of Datasets ................................................................. 31
4.3 Motivating Case Study: Human Activity Recognition .................... 35

5. Autonomous Reconfiguration: Transfer Learning ............................. 40
   5.1 Introduction ................................................................. 40
   5.2 Proposed Framework ...................................................... 42
   5.3 Data Mapping Problem Statement ....................................... 46
   5.4 Signal-level Algorithms .................................................. 48
   5.5 Manifold-based Algorithm ............................................... 55
   5.6 Experimental Evaluation: Signal-level Algorithms ..................... 57
   5.7 Experimental Evaluation: Manifold-based Algorithm ................. 72
   5.8 Summary ................................................................. 75

6. Interactive Reconfiguration: Active Learning ................................. 77
   6.1 Introduction ................................................................. 77
   6.2 System Architecture ...................................................... 79
   6.3 Problem Statement ......................................................... 84
   6.4 Algorithms ................................................................. 87
   6.5 Experimental Setting ...................................................... 95
   6.6 Experimental Validation ................................................ 98
   6.7 Summary ................................................................. 107

   7.1 Introduction ................................................................. 109
   7.2 Problem Statement ......................................................... 111
   7.3 Closed-Loop Deep Activity Recognition ................................ 114
   7.4 Deep Learning Architecture ............................................. 117
   7.5 Experimental Evaluation ................................................ 121
   7.6 Summary ................................................................. 132
8. Energy-Efficient Reconfiguration Management . . . 134
  8.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 134
  8.2 Proposed Framework . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 136
  8.3 Anomaly Screening Unit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139
  8.4 Experimental Validation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 147
  8.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154

9. Conclusion and Future Directions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Statistical features extracted from each axis of accelerometer data</td>
</tr>
<tr>
<td>5.1</td>
<td>Results on network-based clustering for both datasets.</td>
</tr>
<tr>
<td>5.2</td>
<td>Comparison in terms of R-squared value for various data mapping algorithm/regression model combinations</td>
</tr>
<tr>
<td>7.1</td>
<td>Detailed specifications of the used datasets.</td>
</tr>
<tr>
<td>7.2</td>
<td>Comparison of training time of different subjects for each epoch (3D-ConvLSTM vs DeepConvLSTM).</td>
</tr>
<tr>
<td>8.1</td>
<td>Energy consumption of computing individual features.</td>
</tr>
<tr>
<td>8.2</td>
<td>On-body locations used for data collection in normal, and in presence of anomaly, and corresponding accuracy drops.</td>
</tr>
<tr>
<td>8.3</td>
<td>Power saving of feature selection compare to the baseline for various reliability levels ($K \in {1, 2, 3, 4}$).</td>
</tr>
<tr>
<td>8.4</td>
<td>Power consumption and accuracy of different processing units.</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>16</td>
</tr>
<tr>
<td>2.3</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>23</td>
</tr>
<tr>
<td>4.1</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>39</td>
</tr>
<tr>
<td>5.1</td>
<td>42</td>
</tr>
<tr>
<td>5.2</td>
<td>64</td>
</tr>
<tr>
<td>5.3</td>
<td>65</td>
</tr>
<tr>
<td>5.4</td>
<td>66</td>
</tr>
<tr>
<td>5.5</td>
<td>67</td>
</tr>
<tr>
<td>5.6</td>
<td>69</td>
</tr>
<tr>
<td>5.7</td>
<td>70</td>
</tr>
<tr>
<td>5.8</td>
<td>73</td>
</tr>
</tbody>
</table>

1.1 High-level data processing chain for reconfiguration of M-health systems.

2.1 An example showing motifs in a signal on walking data gathered by a sensor.

2.2 Traditional machine learning vs transfer learning.

2.3 A general scenario for active learning.

3.1 Existing approaches categorization for reconfiguration of M-health systems.

4.1 High-level classification of heterogeneity (variation) in M-health monitoring systems.

4.2 Activity recognition accuracy drop due to different causes of data variation.

5.1 An overview of the proposed framework for autonomous reconfiguration of M-health systems.

5.2 Normalized runtime comparison for the three algorithms BFADM, CADM and MADM.

5.3 Scatter plots and corresponding linear regression models for inter-subject and inter-device heterogeneity.

5.4 Scatter plots and corresponding linear regression models for inter-model and inter-placement heterogeneity.

5.5 Impact of similarity threshold $\tau$ on "accuracy versus number of detected pairs".

5.6 Comparison of activity recognition accuracy with and without using transfer learning algorithms (Heterogeneity AR Dataset).

5.7 Comparison of activity recognition accuracy with and without using transfer learning algorithms (REALDISP dataset).

5.8 The data similarity for manifolds in the source and the target domain.
5.9 The accuracy of data labeling. ........................................ 74
5.10 The accuracy of activity recognition with/without MLTL. ......... 74

6.1 High-level illustration of the cost-optimal multi-expert active learning (Co-MEAL) architecture for M-health monitoring systems. ... 80
6.2 Different types of experts in a multi-expert setting in an M-health system. .......................................................... 91

6.3 Comparison between the collaborative active learning algorithm (CAL) and two baseline algorithms: randomly selected samples in active learning (RAL) and a simple transfer learning based approach (STL). The scenario is context change. ................................. 99

6.4 Performance of the collaborative active learning algorithm under a configuration change scenario. ................................. 101
6.5 Performance of the collaborative active learning algorithm under a user need change scenario. ........................................ 102

6.6 Expert uncertainty threshold versus activity recognition accuracy and number of queries (left axes: number of queries, right axes: accuracy). .......................................................... 103
6.7 total number of queries (active training data) versus activity recognition accuracy and number of human queries (left axes: human queries, right axes: accuracy). ........................................ 104
6.8 Number of on-demand experts versus number of queries from human expert(s). ........................................................ 106

6.9 Comparison of non-collaborative and collaborative active learning in terms of total number queries. Each color corresponds to a different expert. The number of queries from imperfect (E2, E3) and perfect (E1) experts decreases in the collaborative learning, because on-demand experts ((a) E4-E7, (b) E4 & E6) increase their knowledge in the previous rounds. ........................................ 107

7.1 High level overview of the proposed active deep learning architecture. 115
7.2 An illustration of the 3D-ConvLSTM architecture. ....................... 118
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3</td>
<td>The 3D input structure with the sliding window representation for data segmentation.</td>
<td>119</td>
</tr>
<tr>
<td>7.4</td>
<td>HAR F1-score of deep learning methods compared to traditional classifiers.</td>
<td>124</td>
</tr>
<tr>
<td>7.5</td>
<td>F1-score for different subjects in the active learning phase versus the number of queries for the Opportunity dataset.</td>
<td>127</td>
</tr>
<tr>
<td>7.6</td>
<td>F1-score for different subjects in the active learning phase versus the number of queries for the Real World dataset.</td>
<td>128</td>
</tr>
<tr>
<td>7.7</td>
<td>The accuracy of activity recognition vs the number of epochs in the training phase.</td>
<td>130</td>
</tr>
<tr>
<td>7.8</td>
<td>The average accuracy for all subjects and the percentage of number of parameters versus the number of frozen layers.</td>
<td>131</td>
</tr>
<tr>
<td>8.1</td>
<td>Current reconfiguration management approaches in networked wearables.</td>
<td>136</td>
</tr>
<tr>
<td>8.2</td>
<td>Energy-efficient uncertainty management framework with the anomaly screening unit (ASU).</td>
<td>137</td>
</tr>
<tr>
<td>8.3</td>
<td>The process of developing the anomaly screening module.</td>
<td>140</td>
</tr>
<tr>
<td>8.4</td>
<td>Accuracy of SUs using DT, kNN, and SVM classifiers respectively.</td>
<td>152</td>
</tr>
<tr>
<td>8.5</td>
<td>Energy consumption comparison of the proposed framework and the conventional approach.</td>
<td>153</td>
</tr>
</tbody>
</table>
Dedication

Dedicated to

my beloved mother, father, and brother
CHAPTER 1. INTRODUCTION

The growing ubiquity of sensor-equipped mobile devices such as pedometers, electroencephalogram (EEGs), smart wristbands, and smartphones has made it possible to capture information about human behavior in real-time. This growth is leading to increased development and deployment of mobile health sensing applications and products [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Nowadays, using mobile devices, it is possible to track step counts, heart rate, calories burned, and even more important information like blood glucose levels for diabetics. Despite the enormous potential of mobile systems, currently existing embedded software for such systems are mostly designed for controlled environments, lab settings, and small trials with fixed and configuration-specific protocols. Scaling these systems up and extending their applications in real-world environments brings about new challenges. We need to design advanced algorithms that accurately analyze data in dynamically changing environments of mobile devices.

1.1 Mobile Health Monitoring

Mobile health (M-health), which the World Health Organization describes as "medical and public health practice supported by mobile devices" [11], is increasingly permeating modern healthcare. By mobile devices we mean smartphones, wearables, and implantable chips that are able to collect data (e.g. sensor data, questionnaire data)
and deliver health-care and preventive-care services. Recent studies show that 83% of physicians in the U.S. use mobile devices to provide care for their patients [12, 13].

More broadly, health-care providers use M-health technology to accomplish a variety of goals: access clinical information of patients, communicate with patients, collaborate with other care providers, and perform real-time health monitoring. Conversely, patients/users use mobile devices to track their own health-data, access clinical records through online portals, and provide feedback to health-care providers (e.g. daily questionnaires) [13, 14, 15, 9]. This makes mobile devices a powerful gateway for improving the quality and effectiveness of services provided to especially the elderly, patients with chronic conditions, and those needing constant monitoring [16, 4, 7, 17, 5, 18].

The increasing number of sensors, applications, and volume of available heterogeneous data necessitate to upgrade the current paradigm of embedded software for mobile computing system used for health-care applications. These future embedded software for M-health systems should be able to automate the system configuration, provide personalized services for users, and facilitate collaboration between users, caregivers, and other connected devices.
1.2 Design Requirements

Currently, conventional supervised machine learning algorithms form the core intelligence of embedded software in mobile systems. Despite the enormous potential and growth in adoption of mobile devices, the current paradigms for such systems lack enough flexibility and accuracy whenever the system is deployed in dynamically changing real-world scenarios. While the current adoption rate of wearables is growing, most consumers lose interest within 30 days because of predefined installation protocols, maintenance issues, and intrusion in users’ daily life [19, 20, 21].

Next generation mobile monitoring platforms need to be adaptive and capable of reconfiguring their machine learning model(s). The conventional approach to dealing with changing environments is to re-train the machine learning algorithm(s) for the new environment. However, collecting new labeled training data is a process that is known to be time-consuming, labor-intensive, and expensive [22, 23, 24, 25, 26]. A better and more cost-effective approach is to leverage related knowledge (e.g. training data for similar contexts) to build an acceptable machine learning model and enhance the model by acquiring context-specific data from the set of experts.

To reach this goal, we need to revisit different components of the general machine learning process chain and equip the system with novel submodules for reconfiguration. We propose a high level process chain for such reconfigurable systems as shown
in Fig. 1.1. It is necessary to apply reconfiguration in different levels of the process chain in order to analyze data accurately. This process chain includes six different units: 1) real-time execution, 2) reconfiguration management, 3) data preparation, 4) model selection, 5) autonomous reconfiguration, and 6) interactive reconfiguration. Based on the required changes, the system may need to move back and forth between different units for adjustment. The real-time execution unit is responsible for online data processing which is similar to conventional supervised learning approach. The reconfiguration management unit, coupled with real-time execution, tracks anomalies and changes in the status of the system. This unit initiates the reconfiguration whenever it detects any variation in context, configuration, and/or user need. Depending on the case, some or all the of other units are activated by the reconfiguration man-
agement unit. In other words, the reconfiguration management unit controls the way that it is required to upgrade the real-time execution machine learning model(s).

The architecture should be able to organize all the available knowledge including labeled training data, unlabeled data, and context data. Each section of the available knowledge may be used in some or all units if it is required. In the data preparation unit, the system should be able to take appropriate actions to prepare data according to the current operating situation. For example, if we are using a device with a higher sampling frequency, we need to apply interpolation techniques on the training data as the first step of reconfiguration. Other adaptation that may be needed is to select best set of features, data normalization, data filtering, and data cleaning.

The model selection unit determines the requirements of the system based on the complexity of situation. For example, if it is required to detect complex set of activities in a human activity recognition (HAR) system, we may need to select an appropriate set of sensors, or learn situation-specific features. Other situations like moving mobile devices, time dependency between data instances effect the actions taken by the model selection unit.

Furthermore, we require autonomous reconfiguration to further adapt the training data for the current situation if needed. The autonomous reconfiguration unit is responsible to design algorithms for mapping data from current situation to another related but different case. The data mapping problem can be simple or highly complex
depending on the complexity of the situation. The goal is to create new situation-
specific labeled data if possible. Depending on the closeness of unlabeled data and
available labeled training data, the autonomous reconfiguration unit interacts with
the model selection unit and the interactive reconfiguration unit. Lastly, the system
may need to gain more knowledge from outside sources to upgrade the system. This
case can happen when it is impossible to reconfigure the algorithms solely using the
labeled training data and its similarity with unlabeled data. Therefore, the system
needs to collect more information from other connected products, users, and caregivers
in order to reconfigure the machine learning models.

To build the aforementioned units, there are several important criteria related
to mobile devices that we need to consider in the design process. First, we need to
design robust and reliable machine learning algorithms in the first place in order to
minimize the reconfiguration cost. Secondly, the system should be able to perform
the adaptation in real-time while the upgraded algorithm is accurate enough in pre-
pdicting events of interest. Third, we need to be able to design low complexity and
energy-efficient algorithms for units that are consistently running on mobile devices.
Finally, we need to minimize the interaction of the mobile system with users, con-
connected devices, caregivers, and physicians for the purpose of system reconfiguration.
In order to achieve these goals, in this dissertation, we tackle fundamental issues in
the design process of the conceptual process chain we presented in Fig.1.1.
1.3 Thesis Contribution

The objective of this research is to develop efficient machine learning algorithms for adaptive and autonomous design platforms in M-health systems. We organize our contribution under four areas.

Contribution Area 1. Autonomous Reconfiguration

- We propose a general framework for autonomous reconfiguration. At a high-level, the framework comprises two states, real-time execution and reconfiguration. For reconfiguration, we develop two different approaches to manage related knowledge and generate newly labeled data for the current situation.

- We propose three signal-level algorithms for applications susceptible to slight changes in data. The main challenge here is that even slight variations in data decrease the accuracy of the system. The first and second algorithms use signal-level similarity measures, while the third algorithm is based on discovery and use of signal motifs, instead of relying on signal similarity. We study the three algorithms and present the trade-offs they offer.

- We propose a manifold-based algorithm for applications where the dataset is complex with lengthy events of interest. We use a manifold learning method to find a low-dimensional representation of data. Considering the periodic behavior of data in many M-health applications, we build trajectories for events in sensor data.

- We assess efficacy of signal-level algorithms using human real activity data with similar classes of data. We show that the accuracy of a human activity recognition (HAR) increases by up to 15% and 28% for different datasets. We analyze the computational complexity of all three algorithms.

- We assess efficacy of the fourth algorithm using a human activity dataset containing complex daily activities. We experimentally show that the accuracy of activity recognition increases by up to 40%.
Contribution Area 2. Multi-Expert Management for M-Health

- We propose a novel architecture designed for multi-expert M-health systems. The key idea is to keep the systems uncertainty below a pre-specified threshold while minimizing the overall cost of re-training the learning algorithm.

- We propose two different algorithms. The first algorithm utilizes the labeled training data to initialize the learner. The second algorithm manages how we are going to select an expert to annotate unlabeled data instances.

- We propose an expert categorization in order to design better algorithms. To this end, we design experiments to simulate real-world experts in an active learning phase. We also design experiments to simulate different types of variation in M-health monitoring systems.

- We evaluate the proposed architecture using two different human physical activity datasets as case studies. We show that using the architecture increases accuracy of activity recognition by up to 45% for the daily activities and by up to 35% for sport activities compared to the case where the architecture is not used. We also show that the number of queries from costly experts is reduced by 78%.

Contribution Area 3. Closed-Loop Deep Learning Reconfiguration

- We propose an active deep learning architecture for HAR systems. The architecture learns representations from related data, and captures the time dependency between consecutive instances. Inspired by RGB representation of images, we add a notion of depth, corresponding to the X, Y and Z components of each sensor type.

- This architecture is also equipped with an active learning submodule to personalize the model for a new situation. A key idea of the architecture is to minimize overall cost of personalization by exploiting the related label data and an effective query strategy.
• We measure the performance of our architecture on two different datasets where the first one includes complex daily activities, while the second one includes simple movements and body postures. Our novel architecture increases the HAR accuracy by up to 25% compared to state-of-the-art models. We also show that by only less than 200 queries the accuracy reaches to the upper bound accuracy for a new situation. Our architecture training time is less than one order of magnitude compared to state-of-the-art deep architectures.

Contribution Area 4. Efficient Reconfiguration Management

• We propose a novel energy-efficient reconfiguration management unit that uses computationally simple screening units for detection of sparse anomalies (i.e. configuration change). This architecture is inspired by power management techniques in design of digital systems.

• We introduce a graph model to capture the importance of features and their correlations. We pose an optimization problem, based on our graph model, to find minimum-cost feature set capable of predicting anomalies within the network.

• We propose a greedy algorithm to solve the minimum-cost feature selection problem as the core intelligence of screening units for network of mobile devices. We analyze the computational complexity of the greedy algorithm.

• We validate the efficiency of our approach using real data collected with motion sensors for physical activity monitoring. Our architecture reduces the energy consumption by up to 90% where the rate of anomalies is limited to 10%.

1.4 Publications

The list of publications related to the above contribution areas are as follows.
Contribution Area 1. Autonomous Reconfiguration


Contribution Area 2. Multi-Expert Management for M-Health


Contribution Area 3. Closed-Loop Deep Learning Reconfiguration


Contribution Area 4. Efficient Reconfiguration Management


1.5 Organization of The Thesis

The rest of the thesis is organized as follows. Chapter 2 to Chapter 4 provide background and motivation. In Chapter 2, we introduce necessary machine learning background along with signal processing techniques. In Chapter 3, we provide a systematic categorization of related research. In Chapter 4, we provide a case study motivating the need for reconfiguration of M-health systems. Chapter 5 to Chapter 8 are devoted to Contribution Areas 1 to 4 outlined earlier. Finally, Chapter 9 provides conclusion and outlines future directions.
CHAPTER 2. PRELIMINARIES

In this chapter, we review a few basic concepts around advanced machine learning approaches that can be used as organizing principles for reconfiguration of embedded softwares in M-health monitoring systems. We begin by formal definitions for signal-level processing techniques that we used to extract knowledge. Then, we have an overview on transfer learning, active learning and deep learning.

2.1 Signal Processing

Mobile devices collect raw data from sensors constantly. The raw data includes long time-series data which will be used in both the training and testing phases of a specific application. A time series is a sequence $S = (s_1, s_2, \ldots, s_l)$, where each $s_i$ is a real number. To extract meaningful information from the sensor data, we need to segment the time series into subsequences. Each subsequence corresponds to a data point (e.g., a specific activity type in HAR).

There are many subsequences in a given source domain. Of these, we need to identify those subsequences that are best related to the target domain. To identify such subsequences we need signal similarity measures to quantify the relatedness. There exist different similarity measures in the literature, including cross-correlation, cosine similarity, and dynamic time warping (DTW). In this work, we use cross-correlation, which we formally define below.
Definition 1 (Cross Correlation). Given two time series $S_1$ and $S_2$, the cross-correlation between $S_1$ and $S_2$ is a vector where each entry is the inner-product of $S_1$ and a given lag (shift) of $S_2$.

The motivation for considering different lags of $S_2$ is to search for the best possible alignment between the two time series. Typically, the maximum value in the cross-correlation vector is reported as the cross-correlation value for the two time series. For better interpretability, the time series are first normalized in such a way that the autocorrelation at zero lag is equal to one. The result is referred to as normalized cross-correlation (NCC). NCC always lies between $-1$ and 1. The extreme case $NCC = 1$ means that the two time series are completely correlated; $NCC = -1$ indicates that the two time series are reversely correlated; and $NCC = 0$ shows that they are totally uncorrelated.

The large size of incoming data for applications of mobile devices (e.g. activity recognition) and having noisy signals due to inherent inaccuracy of sensors inspired researchers to design algorithms for detecting repeatedly occurring subsequences in time series [27, 28]. These subsequences are called signal motifs and are widely used in time series data mining algorithms. Below is a formal definition of signal motif.

Definition 2 (Signal Motif). Given a time series $S$ of length $l$, an integer $p \ll l$, and a predefined distance threshold $d_{th}$, a motif of length $p$ is the subsequence that appears most frequently in the time series $S$. Here, two subsequences are considered
Figure 2.1: An example showing motifs in a signal on walking data gathered by a sensor. The signal has three axes (X, Y and Z). The motif in each axis is the blue portion.

the same if the euclidean distance between them is at most $d_{th}$.

Sometimes motifs other than the most frequent one are also of interest. In general, the subsequence that appears the $k^{th}$ most frequent time in the time series is called $k$-motif. With this denotation, motif and 1-motif are synonyms.

Figure 2.1 shows an example of motifs in a signal representing data on the activity walking. The signal shown in the example is part of a time series of length 4 minutes (48000 samples). The length of the motif in each of the three axes (shown in blue in the picture) is about 4 seconds of data (800 samples). Clearly, the motif length is orders of magnitude shorter compared to the length of the entire signal, motivat-
ing the potential utility of signal motifs in accelerating signal processing algorithms. In particular, most time series data mining algorithms typically have high runtime complexity and similarity search methods in such algorithms can become bottlenecks for classification. Signal motifs reduce computational cost in terms of both time and memory, since there would be no need to search or save the entire data. Signal motifs serve in as sense as a “signature” of the signal source. In Chapter 5, we use this measures to extract knowledge from signals collected using mobile devices.

2.2 Transfer Learning

A common assumption in supervised machine learning algorithms is that the training and testing data are in the same feature space with the same distribution [29]. However, in real-world applications, we may have different feature spaces and/or distributions. When faced with a new situation, recognizing and applying relevant knowledge gained in a related earlier situation is clearly profitable—humans do this rather routinely. Intuitively, transfer learning follows a similar idea. It aims at developing methods to transfer knowledge gained in a source task to improve learning in a related target task. We refer to the training domain where labeled data is available as the source domain, and to the test domain where labeled data is unavailable or very little is available as the target domain [30]. Fig. 2.2 illustrates the difference between transfer learning and traditional machine learning. We formally define transfer learning and a few associated notions in the remainder of this subsection.
Definition 3 (Task). Given a domain, $D = \{X, P(X)\}$, where $X$ is a vector in feature space denoting the input and $P(X)$ is the distribution of $X$, a task consists of two components, a label space $Y$ and a prediction model $M(X)$, and is expressed as $\mathcal{T} = \{Y, M(X)\}$.

The model $M$ is used to predict the label of a new instance $x$ drawn from $X$.

Definition 4 (Transfer Learning). Given a source domain $D_s$ and learning task $T_s$, and a target domain $D_t$ and learning task $T_t$, transfer learning aims to improve the accuracy of the target predictive model $M_t$ in $D_t$ using the knowledge in $D_t$ and $D_s$ in the case where $D_s \neq D_t$ and/or $T_s \neq T_t$.

A specific case where we need transfer learning is when the data distribution in
our target domain is different from that in our source domain. This problem falls into a subcategory called *transductive transfer learning*. In particular, transductive transfer learning corresponds to the case where $D_s \neq D_t$ and $T_s = T_t$.

In transfer learning, as noted in the studies [29, 31, 32], there are three main questions to answer: (1) what to transfer; (2) how to transfer; (3) when to transfer. To find answers to these questions, we need to extract application-specific metrics to measure the quality of available knowledge. Furthermore, we need to develop algorithms to transfer the knowledge in a way that increases the performance of the target domain. Finally, we need to notify the system whenever it is necessary to transfer the available knowledge.

### 2.3 Active Learning

The key idea behind active learning is to allow the learning algorithm to choose the data from which it learns. However, the system should strive to minimize the cost of querying while maintaining the system’s accuracy [33, 23]. There are situations in which unlabeled data is abundant but manually labeling is expensive. In such a scenario, learning algorithms can actively query from the experts for labels. Fig. 2.3 shows a general overview for an active learning scenario. In active learning, a learner often begins with a small number of instances in the labeled training set. However, an M-health system quite often may have to start with no labeled data at all for the current context [34]. This requires more sophisticated strategies in the initialization
phase of the active learning process. The system can benefits from related training data and initialize the learner using appropriate transfer learning approaches.

In order to reduce annotation cost, we need to first come up with a query strategy to select the best subset of unlabeled pool of data instances. The decision of whether or not to query an instance can be framed in several ways. One approach is to evaluate instances using some measure that shows how informative an instance is, since more informative instances are more likely to be queried \cite{35}. However, only considering informativeness of instances is prone to querying outliers.

The least certain instance belongs to the classification boundaries, but is not a good representative of the distribution, so knowing its label is unlikely to improve accuracy on the data as a whole. This phenomenon can occur with sequence labeling tasks as well as with classification \cite{33}. A better method is information density (ID),
proposed in [33], to decrease the chance of selecting outliers. ID is defined as follows.

\[ \phi^{ID}(x) = \phi^{SE}(x) \times \left( \frac{1}{|X_U|} \sum_{x_u \in X_U} \text{sim}(x, x_u) \right)^{\beta} \]  

(2.1)

Here, the informativeness of \( x \) is weighted by its average similarity to all other instances in the unlabeled set \( X_U \), subject to a parameter \( \beta \) that controls the relative importance of the density term. In the formulation presented above, sequence entropy \( \phi^{SE} \) measures the base informativeness. This density measure requires us to compute the similarity of two instances. We measure the similarity of two instances using cosine similarity as follows.

\[ \text{sim}_{\text{cos}}(\vec{x}, \vec{x}_u) = \frac{\vec{x} \cdot \vec{x}_u}{\|\vec{x}\| \cdot \|\vec{x}_u\|} \]  

(2.2)

One potential drawback of ID is that the number of required similarity calculations grows quadratically with the number of instances in \( X_U \). However, these densities are only needed for a limited number of data instances in each active learning phase. Thus, when employing information density in a real-world interactive M-health systems, the density scores are computed once and cached in the back-end server during the actual active learning process.

Furthermore, in real-world health-care applications, labels may come from various sources, including different health-care experts, other mobile devices or even a smart environment with different levels of uncertainty and different cost [36]. Therefore, it
is needed to have an expert management unit that is able to direct the queries to appropriate experts with minimum cost [23, 37].

2.4 Deep Learning

Deep learning is a class of machine learning models with appealing properties for complex domains such as mobile systems. The benefits of deep learning architectures are twofolds in design of M-health monitoring systems. First, we can learn a better representation of data to build generalized models [38, 39, 40]. Learned representations often result in much better performance specifically for tasks with heterogeneous data. Secondly, we can use deep architectures to capture time dependency between consecutive samples in time series. Specifically, Convolutional Networks (CNNs) and LSTM networks are of interest for the above purposes.

**Convolutional Neural Networks (CNNs).** CNNs can capture local dependencies in sensor data—the nearby readings are likely to be correlated [41, 42, 43]. The CNN layers preserve invariant feature scales which result in more general representation of time series data. CNNs learns high level features that are useful for different settings or tasks, corresponding to the underlying factors that appear in more than one setting. A high-level abstraction of data could be used to design robust prediction models in the case of data variation. However, it is assumed that each input is independent of the rest. This has its limitation for sensor data since the problems in this domain deal with multivariate time-series data.
**Long Short-Term Memory (LSTM) Networks.** Recurrent neural networks (RNNs) overcome the limitation of CNNs by using the activation at time $t - 1$ in computing the activation at time $t$. LSTM, a specific type of recurrent neural networks, is suitable for applications where there are very long time lags of unknown sizes between important events. To do so, LSTMs exploit new sources of information so that data can be stored in, written to, or read from a node at each step [44, 42]. During the training, the network learns what to store and when to allow reading/writing in order to minimize the classification errors. Deep networks can be designed using both CNN and LSTM layers in a hybrid setting. First, the CNN layers learns different layers of abstraction from data. Then, the LSTM layers then take over to capture the temporal dependence of activity data.
CHAPTER 3. EXISTING APPROACHES FOR DEALING WITH DATA VARIATION

A variety of approaches have been proposed in recent years to deal with signal heterogeneity in mobile systems. Broadly, we can group the approaches in three categories: reconfigurable supervised learning approaches, robust supervised learning approaches and semi/unsupervised learning approaches as shown in Fig. 3.1. Reconfigurable approaches update a supervised machine learning algorithm using available knowledge, while robust methods design the system to be robust to signal variation in the first place. Semi/unsupervised methods build machine learning algorithms based on limited or no labeled training data. The advanced machine learning approaches as we reviewed in Chapter 2 are used to develop different approaches as it is shown in Fig. 3.1. We review in this section existing works under these three categories.

3.1 Reconfigurable Learning Approaches

In the dynamic environment of mobile devices, it is impossible to collect training data for every single situation. One class of existing approaches deals with this by developing methods that rely on using related knowledge [45, 26, 25, 46, 47] or methods that rebuild a model in such a way that it fits to the current context [48, 49, 45, 50].
Dai et al. [50] present a general boosting algorithm to enable transferring knowledge from one domain to another domain. The algorithm is an extension of an adaptive boosting algorithm [51], and uses training data from the source domain and incrementally builds up the training data for the target domain.

Fallahzadeh et al. [25] propose a network-based approach for cross-subject knowledge transfer in an activity recognition system. The algorithm leverages the feature space cross-subject similarities by constructing a network for the data in source and target subjects and performs community detection to extract core observations in the target data. The work only covers subject-related heterogeneity. Rokni et al. [47]
present a multi-view learning approach for wearables with multiple sensors in which one of the sensors acts as the source domain. The authors show that by using an additional wearable node, performance can be increased by up to 15%. However, the approach needs context-specific training data for the teacher sensor.

Zheng et. al. [26] developed a learning method to find a Web-based similarity function to map activity data in two domains. The sensor data is assumed to be from the same feature space. Based on the learned similarity measures, the cross-domain knowledge transfer task is performed with different confidence level.

Feature mapping is another way for domain adaptation, where the criterion for selecting features is to minimize an approximate distance function between the distributions in the target domain and source domain [48]. Kyle et. al. [45] proposed three heterogeneous transfer learning techniques that reverse the typical transfer model and map the target feature space to the source feature space. They apply the approaches to activity recognition in a smart home context. These approaches use transfer learning at features-level. Chapter 5 falls into this category.

3.2 Robust Supervised Learning Approaches

A drawback of reconfigurable supervised approaches is that it is assumed that always there is relation between source and target context. An alternative approach to deal with changing environments is to design wearable systems in such a way that the machine learning algorithms are robust to the change in the context of the system.
This is achieved in a few different ways: by developing robust algorithms [52, 24, 53, 54], by collecting more training data [55, 56, 57], or by adding redundancy (e.g. more sensors) to the system [58, 59].

In [52], a set of heuristics is presented for activity recognition with sensors displaced within a single body segment. The authors show that within certain limits and with modest quality degradation, activity recognition is tolerant to displacement. Sagha et. al. [24] proposed a probabilistic method to label missing data based on conditional Gaussian distribution. They exploit the correlation among classifier outputs to infer missing values in a probabilistic manner.

Forster et. al. [53] proposed a genetic programming-based feature selection algorithm. The goal is to find a set of discriminative and variation tolerant features that also reduce the energy requirements of the system. The results showed an average 80% of activity recognition accuracy. In [55], the authors present an on-body sensor localization for seven different locations in wearable sensors. It is assumed that for each location, there is an activity recognition model that collects training data for the model. Another location-aware approach is presented in [56]. However, in both works, data is collected for every location, which means large amount of data for training is collected. Furthermore, the approaches rely on models that are trained only for the source domain (e.g. the current subject, the current device).

Allan et. al. investigate sensor heterogeneity in activity recognition systems
A clustering-based method is proposed to categorize different devices and then perform activity recognition using a specific computational model for each category. This approach needs to collect training data for each device category. Zappi et al. [58] investigate the effects of exploiting redundant sensors distributed on the body to use a subset of sensors in the presence of disorientation. They showed that the classification accuracy can be improved to 80% with just three sensors and up to 98% with all the 57 sensors they had in the experiment. Zhu et al. [59] study the usefulness of different features in a multi-sensor activity recognition system for robust signal processing. They evaluate several types of features and propose different normalization approaches.

In [60], a deep architecture only using Convolutional Neural Networks (CNN) is proposed to capture local dependency and mitigate the effects of variation. They also exploit a modified weight sharing technique for the accelerometer data to get further improvements. In another work [42], a deep architecture for HAR based on Convolutional and LSTM recurrent units is proposed for multi-sensor wearable systems. The advantage of this work is that it explicitly models the temporal dynamics of activity data in addition to representation learning. Chapter 7 and Chapter 8 fall into this category.
3.3 Semi/Unsupervised Learning Approaches

Unsupervised learning algorithms are a good choice for non-critical applications or unknown category of data in the lifetime of the system. Vahdatpour et. al. [28] propose a method for unsupervised discovery of abnormal occurrences of activities in multi-dimensional time series data. They use the random projection algorithm as their motif discovery algorithm [61]. They show the efficacy of this approach on two different datasets.

A space-saving unsupervised algorithm for systems with stringent resources collecting streaming data is presented in [62]. The space-saving algorithm only holds the top-$K$ motifs in the memory of mobile device for some value $K$ and discard the low frequent patterns through the process.

Semi-supervised learning algorithms are designed to use available training data in addition to unlabeled data. Stikic et. al. [63] introduce a semi-supervised algorithm that combines small amounts of labeled data with easily obtainable unlabeled data. Their method propagates information through a graph that contains both labeled and unlabeled data.

Researches have also used active learning to supplement semi-supervised learning. Typically, an active learning approach queries experts (e.g. human annotator) to label data. Longstaff et. al. [64] investigated the applicability of active learning for activity recognition and showed that using active learning improves accuracy by up
to 10% compared to a semi-supervised learning that does not use active learning. In [34], a multiple-expert active learning architecture is proposed for developing efficient machine learning models for a wearable system. In particular, a wearable device in the architecture learns from experts available to the system, while minimizing the cost of data labeling. Chapter 6 falls into this category.
CHAPTER 4. MOTIVATION FOR RECONFIGURATION

In this chapter, we introduce different causes of data variation in M-health systems. Then, we have an overview on datasets we used in this dissertation. Finally, to motivate the need for reconfiguration more concretely, using the first two datasets, we demonstrate the negative effect of various data variations on the performance of human activity recognition (HAR) systems.

4.1 Scenarios Driving Reconfiguration

Before presenting the details of the case study, we review in this section the different causes of data variation (heterogeneity) in M-health systems. We depict in Fig. 4.1 a hierarchical categorization of the variations. At the higher level, the variations can be grouped in three classes: configuration change, context change, and user demand change.

Configuration change. Mobile devices are often built using low cost sensors, which are inaccurate and poorly calibrated. As a result, the signal readings are noisy and with low sensitivity. On the other hand, activity recognition models are built based on training data collected on commercial platforms, which consist of accurate and dedicated standalone sensors with higher sampling frequency. Furthermore, it is possible to consider a scenario where sensors leave or join the M-health system. We call this
Figure 4.1: High-level classification of heterogeneity (variation) in M-health monitoring systems.

hardware-related heterogeneity **configuration change**. Depending on the complexity of the situation, both autonomous and interactive reconfiguration can be applied to deal with this type of heterogeneity. In this dissertation, we focus on the variant of configuration changes resulting from addition of sensory devices, sensor calibration, and sampling frequency.

**Context change.** Sensor data generated during personalized exercises or tasks performed by one subject (or patient) may be too specific or inadequate to be used as a training set for a new subject. Even for the same subject, physiological patterns may change over time. In another related situation, subjects may install their mobile devices in different locations or orientations, which leads to inaccurate classification. Furthermore, any change related to a subject’s life style may be categorized under this type of variation. In this dissertation, we focus on inter-subject, misplacement, and displacement in different chapters.
**User need change.** With the emergence of IoT platforms, M-health monitoring systems need to adapt whenever a user’s preferences change. For example, a user may want to enhance the precision of the system as they change their life style. Furthermore, a user may need to increase the capability of the system. For example, a user may need to classify new set of activities due to changes in their workout program. We refer to any user-initiated changes as *user need changes*. In this work, we consider the case where a user adds new classes of data to recognize by the M-health system. Due to the complexity of this situation, we almost always need to perform interactive reconfiguration unless the training labeled data pool includes related databases for autonomous reconfiguration.

### 4.2 Description of Datasets

In this section, we have an overview on datasets we used in this dissertation. These datasets are used in the following case study to justify the need for reconfiguration and for validations of different approaches in later chapters. Based on the problem setting, we choose one or several datasets in each chapter.

#### 4.2.1 Heterogeneity HAR Dataset

Nine subjects were asked to perform six different daily activities: ‘Walking’, ‘Sitting’, ‘Standing’, ‘Stair down’, ‘Stair up’, and ‘Biking’. Subjects were asked to perform
the activity set while carrying eight smartphones (two instances of four device types: LG Nexus 4, Samsung Galaxy S+, Samsung Galaxy S3, and S3 mini) on their waist. Each smartphone is equipped with a tri-axial accelerometer. The maximum sampling frequency for LG Nexus 4, Samsung Galaxy S+, Samsung Galaxy S3, and Samsung Galaxy S3 mini is 200 Hz, 150 Hz, 100 Hz, and 50 Hz respectively [57]. This dataset is publicly available at the University of California, Irvine (UCI) Machine Learning Repository.

4.2.2 REALDISP HAR Dataset

This dataset includes a wide range of physical activities (warm up, cool down and fitness exercises), sensor modalities and participants (17 subjects) with sampling frequency of 50Hz [65]. The REAListic sensor DISPlacement dataset builds on the concept of ideal-placement, self-placement and induced-displacement. Self-placement reflects a user’s perception of how sensors could be attached, while ideal placement mirrors the predefined data collection protocol. This dataset is used to resemble the misplacement scenario for mobile devices. This dataset is also publicly available at the UCI Machine Learning Repository.

4.2.3 Opportunity Dataset

The Opportunity Activity dataset concerns human activities [66]. The dataset contains data collected from sensors configured on four subjects who perform Activities
of Daily Living (ADL) (e.g., drink from cup, open door, close door, open dishwasher).

There are 18 classes in this activity recognition task. The Null class refers to either non-relevant activities or non-activities. The full setup includes both ambient and on-body sensors. However, here we only consider tri-axial wearable sensors as our focus is on M-health systems. The sensor set includes accelerometers, gyroscopes, and magnetometers in different locations on the body. More detail about the dataset can be found in [66].

4.2.4 Real World Dataset

This dataset is collected by the Data and Web Science Research Group at the University of Mannheim and is publicly available at their website [67]. Fifteen subjects, seven females and eight males, were asked to perform eight different daily activities: Walking, Running, Sitting, Standing, Stair down, Stair up, Jumping, and Lying. Subjects were asked to perform the activity set while carrying seven mobile devices located on chest, shin, forearm, thigh, upper-arm, head, and waist. Each subject performed each activity for roughly 10 minutes, except for jumping which is performed only for 100 seconds. Each mobile device is equipped with different types of sensors, including tri-axial accelerometer and gyroscopes.
4.2.5  *Daily and Sports Activities Dataset*

The fourth dataset is the "Daily and Sports Activities Dataset" [68] which is publicly available at University of California-Irvine’s data repository. In this dataset, subjects were asked to perform the activity set while sensors are placed on five different body locations. Eight subjects were asked to perform nineteen activities including sport and daily activities (e.g. walking on treadmill, rowing, basketball playing, standing in an elevator still, walking in a parking lot) each for a duration of five minutes. Each device collects data with sampling frequency of 25Hz from tri-axial inertial sensors (i.e. accelerometer, magnetometer, gyroscope).

4.2.6  *30-Movements-18-Activity Dataset*

This dataset includes simple-activities collected from multiple nodes. Each node, which is also called a mote, has a triaxial accelerometer, a biaxial gyroscope. The sampling frequency is 22Hz. TelosB motes is used to measure acceleration and angular velocity of different body segments using a tri-axial accelerometer and a biaxial gyroscope. Three subjects were asked to perform 30 different daily activities such as 'sit to stand', 'sit to lie', 'bend and grasp', 'kneeling', 'step forward', 'step backward', etc. Subjects are asked to put on 18 nodes on their body. We used this dataset to resemble scenarios where the sensors are misplaced or displaced in a M-health monitoring system [69].
4.3 Motivating Case Study: Human Activity Recognition

We, here, investigate the effects of subject variation, device calibration and sensitivity, sampling frequency variation, and misplaced devices on the accuracy of HAR. We used the Heterogeneity and REALDISP HAR datasets. We begin by data preparation, used classification models along with variation scenarios. Then, we present the results of accuracy drop due to the above variations.

4.3.1 Pre-Processing and Data Segmentation

We applied a moving average filter on the activity data to remove high frequency noises from data. Therefore, another external factor is eliminated from the acceleration data. After data filtering, a two-second sliding window with an overlap of 25% is used for data segmentation. The choice of sliding window is based on the period of different activities in the selected datasets.

4.3.2 Feature Extraction

A variety of features can be extracted from human activity data. Previous studies have identified statistical features that are effective for activity recognition [70, 71]. Table 4.1 lists the set of features that we extracted from individual data segments. In addition to the statistical features, we extracted 10 other features which we call morphological features. These features aim to capture structural properties of the signal. The ten features are obtained by taking ten samples that are evenly spaced in
Table 4.1: Statistical features extracted from each axis of accelerometer data.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amp</td>
<td>Amplitude of signal segment</td>
</tr>
<tr>
<td>Med</td>
<td>Median of signal segment</td>
</tr>
<tr>
<td>Mean</td>
<td>Mean value of signal segment</td>
</tr>
<tr>
<td>Max</td>
<td>Maximum amplitude of signal segment</td>
</tr>
<tr>
<td>P2P</td>
<td>Peak to peak amplitude</td>
</tr>
<tr>
<td>Var</td>
<td>Variance of signal segment</td>
</tr>
<tr>
<td>Std</td>
<td>Standard deviation</td>
</tr>
<tr>
<td>RMS</td>
<td>Root mean square power</td>
</tr>
<tr>
<td>S2E</td>
<td>Start to end value</td>
</tr>
</tbody>
</table>

time over the entire signal segment. We should mention that the sensor orientation is the same for training and testing except when we are working with the self-placement scenario for the RERALDISP dataset.

4.3.3 Comparison Scenarios

To show the effects of data heterogeneity on HAR performance, we used different machine learning algorithms to detect physical activities. In particular, we built prediction models using different types of classifiers including decision tree (DT), random forest (RF), and k-nearest neighbor (k-NN) algorithms. We considered four different scenarios to generate training and test data. We discuss the scenarios below starting with the "upper bound" that can be expected to be attained in each case.
• **Upper-bound:** For each domain (refer back to Section 2.2), we split the data using 10-fold cross-validation and measure the classification accuracy. Data with the same context/configuration (i.e. same subject and same device) form a domain. This is an upper-bound on the accuracy that we can get from the available sensory system.

• **Inter-subject:** The model is built based on all the subjects except one, which is used for test. This scenario is included to check inter-subject variation.

• **Inter-model:** we build the classifier using the data of one specific sampling frequency and test it on data with different sampling frequency. This scenario is included to check the effects of sampling frequency on performance of HAR.

• **Inter-device:** for each device type (e.g. Samsung Galaxy S3), we use one of the instances for training and the other for testing. This scenario is included to check signal heterogeneity of the same device type.

• **Inter-placement:** For each subject, we build the classifier using the data of ideal sensor placement and test it on data of self-placement scenario. This scenario is included to check the effects of sensor displacement on performance of HAR.

The Inter-device scenario is used only for the Heterogeneity dataset and the Inter-placement scenario is used only for the REALDISP dataset. The other scenarios are used for both datasets.
4.3.4 Activity Recognition Accuracy

Fig. 4.2 shows the accuracy drop due to different data variation scenarios for the various prediction models we considered. Fig. 4.2 shows that, if we take one subject out, the accuracy drops by 20% for the first dataset and by 30% for the second dataset on average compared to the upper bound. For the inter-model scenario, which depicts the effects of sampling frequency, an accuracy drop between 8% to 25% for both datasets is observed. Fig. 4.2a shows that the accuracy drops by up to 20% compared to upper-bound for inter-device scenario. Fig. 4.2b shows that, if the wearable device is installed by the user, the accuracy drops by up to 12%. For all scenarios, we present the standard deviation value on each bar. Our case study shows that the effect of the aforementioned types of heterogeneity on the accuracy of HAR systems is significant, which suggests a need for design of effective algorithms and frameworks for reconfiguration of computational models in such systems.
Figure 4.2: Activity recognition accuracy drop due to different causes of data variation.
CHAPTER 5. AUTONOMOUS RECONFIGURATION:

TRANSFER LEARNING

The key focus of this chapter is to further go to in details for designing autonomous reconfiguration unit as introduced in Chapter 1. We build several algorithms to be used in different situations. Each algorithms is applicable for a specific case depend on available computational resources, need for real-time reconfiguration, nature of data, and types of data variation.

5.1 Introduction

We, here, build several algorithms in order to address the below limitations related to the current paradigm of mobile health systems. First, the accuracy of machine learning algorithms drops whenever the system specification or installation changes (e.g. sensor calibration, device orientation, sampling frequency). Second, data collected for one setting (e.g. a specific type or brand of sensor) may no longer be suitable for a new setting. For example, the training data may be collected using high quality sensors while the final data processing may be conducted on a different platform such as smart-phones and smart-watches. The above scenarios can be categorized as configuration change. Third, sensor data, generated during personalized exercises or tasks, may be too specific or inadequate to be used as a training set for a new
subject/patient. Moreover, even for the same subject, the physiological patterns may change over time. Inter-subject differences are identified as context change (referring back to Section 4.1).

Within this broad aim, our key focus is to design algorithms for mapping data from one setting to another related but different setting. However, accurate inter-device and inter-subject data mapping need different approaches based on the intrinsic of data. Therefore, depending on the application, we need to come up with different solutions. We list different types of data variation scenarios as follows: 1) Different classes of data are similar and, even a slight shift in the distribution of data leads to inaccurate data mapping between different domains. Feature space representation of data, also, intensify the effects of variation related to speed (e.g. walking with different paces) and amplitude (e.g. sensor calibration) of different settings for some events of interest; Many critical applications such as heart-failure patient monitoring or fault detection systems are belongs to this category; and 2) The events of interest contain longer and more complex patterns and we need to cope with data in feature level (e.g. complex set of activities). These types of data are more robust against small changes, and it is better to cope with data after removing the effects of intra-class variations. For this type of data, we can work on feature-level data instead of signal-level data.

In the rest of this chapter, we present the general proposed framework for au-
Figure 5.1: An overview of the proposed framework for autonomous reconfiguration of M-health systems.

We propose two different approaches for solving the data mapping problem to be used in the aforementioned data variation cases in Section 5.4 and Section 5.5. Then, in Section 5.6 and Section 5.7, using human activity recognition data, we evaluate the performance of the two different approaches.

5.2 Proposed Framework

We present in this section the framework we propose to support autonomous reconfiguration of wearable systems. A high-level illustration of the framework is given in Fig. 5.1. The framework consists of two main “states” (to borrow a term from finite automata): real-time execution and reconfiguration.
The real-time execution state comprises the main signal processing chain and a data variation detection unit. The reconfiguration state uses a knowledge transfer submodule to find the relation between the target domain and the source domain. We discuss the details of the major elements of the framework in the remainder of this section.

5.2.1 Real-time Execution

The real-time execution state runs to detect the events of interest for the application (e.g. activity type). Sensors in the mobile system collect data from human body. Preprocessing techniques are performed on the sensor raw-data, mostly for noise filtering, normalization and interpolation. After preprocessing, data segmentation is performed to divide up data streams into segments corresponding to events of interest. The next step maps—using a data mapping model coming from the reconfiguration state—subsequences in the target domain to subsequences in the source domain. The data mapping model would work as an identity function when the test data has the same feature space as the training data. Then, features are extracted from the mapped data segments. Finally, a machine learning model is used to detect events of interest in our application.

Alongside the machine learning model, a data variation detection unit determines the presence of signal variation—and if variation is present what type of variation it is—and activates the reconfiguration state if needed. The determination of data
variation can be framed in several ways. One approach is to evaluate the model’s performance using some measure that shows how the system can classify an instance. The data variation detection unit asks for reconfiguration whenever there are instances about which it is uncertain how to label. This approach can be implemented using probabilistic machine learning models or a committee of learning models [72, 34].

5.2.2 Reconfiguration: Transfer Learning

The purpose of the reconfiguration state is to construct the data mapping model for the real-time execution state. The data mapping model is built based on the relation between the target domain and the source domain. The transfer learning submodule finds the corresponding data segments in these two domains. The actual data mapping model generated in the reconfiguration phase can be used in two different directions. If the relation between the source domain and the target domain is obtained by a regression model, we build a function for data mapping. The mapping model reverses the typical transfer model and maps data points from the target domain to the source domain. If the relation is not captured by a single regression function, we generate a new set of labeled data from the point to point mapping and retrain the computational model using the newly labeled data. In extracting knowledge from the source domain, the reconfiguration state uses a set of dictionaries, which is generated off-line.
5.2.3 Reconfiguration: Off-line Data Preparation

The block identified as "off-line data preparation" in Fig. 7.2 refers to the way the available knowledge in the source domain is used within the reconfiguration state. A key component in the offline data preparation is the dictionary, which is used to organize knowledge from the source domain(s). Below, we formally define dictionary, following similar notation as in [73].

**Definition 5** (Dictionary). A dictionary (database) $DB$ is an unordered set of data instances of possibly different lengths. We denote by $DB_i$ the $i^{th}$ data instance in the dictionary. Each data instance $DB_i$ can be a feature vector, a signal motif, a data segment, or a time series depends on knowledge transfer algorithm.

The dictionary is an important construct for several reasons. First, the knowledge stored in the dictionary determines the time and method needed for searching relevant knowledge. Second, the structure and size of the dictionary determine the search method. Therefore, the dictionary needs to be a carefully-chosen, small subset of the source domain(s) data. Referring back to the three main questions transfer learning need to address (Section 2.2), the dictionary answers the how to transfer part. The what to transfer part in our case is addressed using similarity metrics. In this dissertation, we use two different approaches as we will see later in this chapter.
5.3 Data Mapping Problem Statement

In this section our goal is to formulate the data mapping problem needed in the knowledge extraction component in the reconfiguration state. Whenever a user starts using the mobile system, context-specific unlabeled data (target) is collected gradually throughout the system’s life time and may well be different from the available labeled data (source). The goal is to find a model to map data from the target domain to the source domain.

Suppose $DB_T$ is the dictionary of $n$ time series in the target domain, and $DB_S$ is the dictionary of $m$ time series in the source domain. Our problem is to map each time series in $DB_T$ to a time series in $DB_S$. A time series in the target domain and a time series in the source domain may have different lengths. This difference may be related to difference in sampling frequency, user attribute, or length of trial. We are interested in an asynchronous data mapping (ADM) problem, which we formally define below. The term asynchronous is used to indicate that the data in the source domain and the target domain are collected at different times.

**Problem 1** (ADM). *The asynchronous data mapping problem is to find a general mapping model $M : T \rightarrow S$ to map a sample $t \in T$ to a sample $s \in S$ for all instances such that the overall similarity of data mapping is maximized. A sample refers to a raw-data signal sample (in Section 5.4) or a feature vector for a micro-segment (in Section 5.5).*
Samples do not provide much information about the underlying application on their own especially if we work with signal-level data. Even for feature level-data, it is more robust to find related consecutive micro-segments. Therefore, we need to consider data segments instead of samples, as we need to have meaningful information to extract knowledge. Each data segment corresponds to an event in our application.

The ADM problem can be formulated as an optimization problem. Let $\alpha_{ij}$ be a binary variable indicating whether or not time series subsequence $DB_{Ti}$ in the target domain is mapped to a time series $DB_{Sj}$ in the source domain. Let $\lambda_{ij}$ be the magnitude of similarity (i.e. NCC) between $DB_{Ti}$ and $DB_{Sj}$. Our objective then is to maximize the data mapping similarity over all mappings subject to making sure that the similarity is higher than a pre-specified threshold $\tau$. Thus, we have the following optimization problem:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_{ij} \lambda_{ij}$$  \hspace{1cm} (5.1)

subject to: $\lambda_{ij} \geq \tau$ \hspace{1cm} (5.2)

$\alpha_{ij} \in \{0, 1\}, \ \forall \ i, j$ \hspace{1cm} (5.3)

$\sum_{i=1}^{n} \alpha_{ij} = 1, \ \forall \ j \in \{1, 2, \ldots, m\}$ \hspace{1cm} (5.4)

The constraint (5.2) guarantees that the similarity between mapped time series is
greater than a threshold similarity $\tau$. The constraint (5.3) ensures that the variable $\alpha_{ij}$ takes only binary values and the constraint (5.4) implies that only one mapped time series in the source domain should be selected for each subsequence in the target domain.

This is an integer programming problem, which in general is known to be NP-Hard. Rather than solving this problem exactly, we settle for an approximate solution, which we obtain by using greedy approaches that maximize the similarity. In Section 5.4, we work with raw-data signal-level data, while in Section 5.5, we work with feature-level data for data mapping.

5.4 Signal-level Algorithms

We present in this section three different data mapping algorithms we developed. We also present auxiliary procedures used by the algorithms and complexity comparisons.

5.4.1 Pattern Finding Procedure

All three of our data mapping algorithms use a common procedure whose purpose is to find a specified pattern in a given time series. The pattern finding algorithm is outlined in Algorithm 1. The algorithm takes as input a time series $T$, a pattern $pat$, and an integer $overlap$ indicating the overlap between consecutive windows allowed in the search for $pat$ in $T$. The value of $overlap$ ranges between zero and the size of the pattern ($pat.size$). The algorithm returns the subsequence $match$ in $T$ that is the
Algorithm 1 FindPattern

**Input:** $T$, $pat$, overlap

**Output:** match, sim

1. $sim \leftarrow 0$, $start \leftarrow 1$, $stop \leftarrow pat.size$
2. **while** $stop \leq T.size$ **do**
3. $max \leftarrow \text{similarity}(T(start : stop), pat)$
4. **if** $max > sim$ **then**
5. $sim \leftarrow max$
6. $match \leftarrow T(start : stop)$
7. **end if**
8. $start \leftarrow stop - overlap + 1$
9. $stop \leftarrow start + pat.size - 1$
10. **end while**

most similar to $pat$ and the numerical value of the similarity $sim$, which is computed via the subroutine SIMILARITY.

All three of our data mapping algorithms take as part of their input a threshold $\tau$ that is used to decide whether or not two signal segments are considered similar. Two signal subsequences $T = \{t_1, t_2, \ldots, t_l\}$ and $S = \{s_1, s_2, \ldots, s_p\}$ are said to be similar if the NCC between them exceeds the given threshold $\tau$.

5.4.2 The Brute Force ADM Algorithm (BFADM)

Our first data mapping algorithm, outlined in Algorithm 2, is a brute force method. The algorithm takes as input a time series $T = \{T_{seg_1}, \ldots, T_{seg_n}\}$, the dictionary $DB_S$ of the source domain, the overlap allowed $overlap$, and a similarity threshold $\tau$. For each $T_{seg_i}$, Algorithm 2 calls Algorithm 1 to find the closest data segment to $T_{seg_i}$ in the dictionary $DB_S$, and adds it to the data structure $Pairs$ if the similarity value
Algorithm 2 Brute Force ADM Algorithm (BFADM)

Input: $T = \{T_{seg_1}, \ldots, T_{seg_n}\}, DB_S, \text{overlap}, \tau$

Output: Mapping Function $F$

1: Initialize $Pairs \leftarrow \emptyset$
2: for all $T_{seg_i} \in T$ do
3: \hspace{1em} $sim \leftarrow -1$
4: \hspace{2em} for all $DB_{S_j} \in DB_S$ do
5: \hspace{3em} $(m, s) \leftarrow \text{FindPattern}(DB_{S_j}, T_{seg_i}, \text{overlap})$
6: \hspace{3em} if $s > sim$ then
7: \hspace{4em} $(match, sim) \leftarrow (m, s)$
8: \hspace{3em} end if
9: \hspace{2em} end for
10: \hspace{1em} if $sim > \tau$ then
11: \hspace{2em} $Pairs \leftarrow Pairs \cup (T_{seg_i}, match)$
12: \hspace{1em} end if
13: end for
14: $F \leftarrow \text{MappingModel}(Pairs)$

returned by Algorithm 1 exceeds the threshold $\tau$. This is repeated over all the selected segment in the target domain. The elements of $Pairs$ are then used to construct the mapping model $F$, which is returned as the output of the algorithm.

The brute force search employed in Algorithm 2 is accurate, but it can be inefficient in terms of runtime because of the large search space involved. In the next two subsections, we present two other algorithms which are much faster because they work with much reduced search space, at the expense of potentially slightly lower accuracy.
Algorithm 3 Clustering-based ADM Algorithm (CADM)

**Input:** $T = \{T_{\text{seg}1}, \ldots, T_{\text{seg}n}\}$, $DB_S$, overlap, $d_i$, $\tau$

**Output:** Mapping Function $F$

1: Initialize $Pairs \leftarrow \emptyset$
2: $\{S_{\text{seg}1}, \ldots, S_{\text{seg}m}\} \leftarrow DB_S(\text{overlap})$ /*segmentation*/
3: Initialize $G : V \leftarrow \{v_1, v_2, \ldots, v_m\}$, $E \leftarrow \emptyset$
4: for $i = 1$ to $m$ do
5:   for $j = 1$ to $m$ do
6:     $\text{dis} \leftarrow \text{Distance}(S_{\text{seg}i}, S_{\text{seg}j})$
7:     if $\text{dis} < d_i$ then
8:       $E \leftarrow E \cup (v_i, v_j)$
9:     end if
10:   end for
11: end for
12: $C = \{C_1, \ldots, C_z\} \leftarrow \text{CommunityDetection}(G)$
13: Let $c_i \subset C_i$ be representative samples from $C_i$
14: for all $T_{\text{seg}} \in T$ do
15:   $DB_C \leftarrow \text{argmax}_{C_i \in C} \text{similarity}(c_i, T_{\text{seg}})$
16:   $(\text{match}, \text{sim}) \leftarrow \text{FindPattern}(T_{\text{seg}}, DB_C, 0)$
17:   if $\text{sim} > \tau$ then
18:     $Pairs \leftarrow Pairs \cup (T_{\text{seg}}, \text{match})$
19:   end if
20: end for
21: $F \leftarrow \text{MappingModel}(Pairs)$

5.4.3 Clustering-based ADM Algorithm (CADM)

Our second data mapping algorithm reduces the search space using a clustering approach. To find data segments that can be grouped together, we first build a network of data segments, and then apply a community detection method on the resulting network to obtain the clustering. We formalize the approach below.

**Definition 6** (Similarity Network). Given the source domain data segments $DB_S$
= \{ S_{seg_1}, \ldots, S_{seg_m} \}, the similarity network \( G = (V,E) \) is a network where each vertex \( v_i \in V \) corresponds to data segment \( S_{seg_i} \), and \( (v_i, v_j) \in E \) whenever the cosine distance \( (1 - \text{cosine similarity}) \) between data segments \( S_{seg_i} \) and \( S_{seg_j} \) is lower than a pre-specified threshold \( d_t \).

We extract features (Table 4.1) from each data segment and then calculate the cosine distance between every pair. After the similarity network is clustered, we select a small set of representative vertices \( c_i \) from each cluster \( C_i \), for \( i = 1, 2, \ldots, z \), where \( z \) is the number of clusters. These representatives are used to find the pairs needed for building the mapping model. Algorithm 3 gives the details of the algorithm.

As mentioned earlier, the clustering approach reduces the cost of online search for similar pairs, but this comes at the expense of the time spent on building and clustering the similarity network. However, the network building and clustering is performed off-line, making the approach a better choice for real-time knowledge transfer than the brute-force algorithm BFADM.

5.4.4 The Motif-based ADM Algorithm (MADM)

The idea behind our third data mapping algorithm is also to reduce the search space for pairs, but the reduction is achieved in a completely different way—namely, by extracting signal motifs from every time series in the source domain. The algorithm is outlined in Algorithm 4. It takes as input the time series \( T \) from the target domain,
the source domain dictionary $DB_S$, the minimum distance for motif discovery $d_{th}$, and the number $k$ of motifs in each time series, and a similarity threshold $\tau$. Like the previous two algorithms, its output is a mapping model.

There exists number of works for locating known patterns in time series [27, 61, 73]. In our case, we used a variant of the brute force algorithm proposed in [73]. In particular, in Algorithm 4, we extract $k$ motifs for each activity type, and save all of the motifs in a dictionary named $DB_M$. In each iteration of the motif discovery algorithm, we remove the occurrences of detected motifs by using the minimum dis-
distance $d_{th}$. This way, we avoid having similar motifs in the dictionary. The function \textsc{FindMotifs} returns the $k$ motifs for the given distance threshold $d_{th}$ and a given time series $DB_{Si}$. Then, for each $T_{seg_i}$, Algorithm 1 is called to find the closest data segment to $T_{seg_i}$ in the dictionary $DB_M$, which is then added to the data structure \textit{Pairs} if its similarity is greater than the threshold $\tau$. This is repeated until we iterate through all selected signal segments in the target domain.

Although motif discovery in general is a computationally expensive operation, in our framework, the motif discovery phase is carried out off-line and therefore the approach is a great choice for real-time knowledge transfer.

### 5.4.5 Complexity

The complexity of the \textsc{FindPattern} algorithm depends on the complexity of the similarity method used to compare two subsequences. Since we use NCC as a correlation measure, the complexity of \textsc{FindPattern} is $O(nm)$, where $n$ is the length of the time series and $m$ is the length of the subsequence. Next, we contrast the three data mapping algorithms against each other, considering only the on-line portions of the algorithms (i.e., Lines 2–13 of Algorithm 2, Lines 14–20 of Algorithm 3, and Lines 5–16 of Algorithm 4).

The data mapping algorithms have different search spaces due to the differences in the organization of their respective dictionaries. The BFADM algorithm needs to run \textsc{FindPattern} for all the selected data segments in the target domain. Therefore,
Algorithm 5 ML-based Transfer Learning (MLTL)

Input: $X_S = \{X_{11}, \ldots, X_{1k}, X_{21}, \ldots, X_{2k}, \ldots, X_{|S|k}\}$

Input: $X_T = \{X_{T1}, \ldots, X_{Tm}\}$

Output: Mapping Function $F$

1: Initialize $Pairs \leftarrow \emptyset$
2: for all $X_{ij} \in X_S$ do
3:     $M_{ij} \leftarrow \text{LEARNMANIFOLD}(X_{ij})$
4: end for
5: for all $X_{Ti} \in X_T$ do
6:     $M_{close} \leftarrow \text{argmin}_{M_{ij}} \text{DISTMANIFOLD}(X_{Ti}, M_{ij})$
7:     $Pairs \leftarrow Pairs \cup (X_{Ti}, M_{ij})$
8: end for
9: $F \leftarrow \text{MAPPINGMODEL}(Pairs)$

the number of searches involved is $N_T \times |DB_S|$, where $N_T$ is the number of selected subsequences in the target domain and $|DB_S|$ denotes the number of time series in the dictionary $DB_S$. The CADM algorithm needs to run $\text{FINDPATTERN} \ N_T \times |DB_C|$ times, where $DB_C$ is the dictionary of the selected clusters. Finally, the MADM algorithm needs to run the $\text{FINDPATTERN}$ procedure $N_T \times |DB_M|$ times, where $DB_M$ is the dictionary of extracted signal motifs.

From our empirical evaluations (to be discussed Section 5.6), we observed that $|DB_C|$ is typically an order of magnitude smaller than $|DB_S|$ and $|DB_M|$ is typically at least two orders of magnitude smaller than $|DB_S|$.

5.5 Manifold-based Algorithm

We present in this section a manifold learning-based approach suitable for complex datasets with lengthy events of interest. In this algorithm, the dictionary organization
is different compared to the signal-level algorithms (Section 5.4). We use the Locally Linear Embedding (LLE) algorithm to construct activity manifolds from source domain. We chose LLE over other non-linear manifold learning (e.g. Isomap) methods because it has lower complexity. Non-linear manifold learning approaches are able to detect the non-linear nature of activities. We construct manifolds for each class of data separately in a supervised manner. The number of manifolds for each domain is equal to the number of classes (i.e. labels).

Let \( X_{ij} \) be a set which includes all data segments for the source domain \( i \) in feature space with activity label \( j \). Suppose each instance of data is a \( q \)-dimensional vector, where \( q \) is the number of features. Let \( M_{ij} \) be the manifold we learned from \( X_{ij} \). The number of manifolds is equal to \( |S| \times k \) where \( |S| \) is the number of source domains. Let \( X_{T} = \{X_{T1}, X_{T2}, \ldots, X_{Tm}\} \) be the set of unlabeled data instances from the target subject. Each \( X_{Ti} \) is then segmented (400 ms with 50% overlap), and subsequently transformed to the feature space. We call each segment micro event because each one corresponds to a part of the event we want to detect. The set of consecutive micro-segments form a time-series corresponds to the ADM problem in Section 5.3.

The manifold-learning algorithm is outlined in Algorithm 5. The algorithm takes as input the data for the source domain(s) \( X_S \), the data for the target domain \( X_T \), and the label set \( L \) consisting of \( k \) classes. The algorithm finds \( k \) manifolds for each source domain by calling the routine LEARNMANIFOLD (an LLE algorithm). It then
searches to find the closest manifold to each $X_{Ti}$ in the target domain. The function $\text{DistManifold}$ maps $X_{Ti}$ into manifold $M_{ij}$ and then measures the distance. We used Hausdorff metric to measure the closeness of a manifold to the mapped data [74]:

$$DH(X_t, X_s) = \frac{1}{|X_t|} \sum_{x_t \in X_t} \min_{x_s \in X_s} ||x_t - x_s||$$

(5.5)

Because the number of points in the target domain is low, the distance measurement needs to be modified to guarantee symmetry. Therefore we define the final distance as follows:

$$\text{DistManifold} = \max \{ DH(X_t, X_s), DH(X_s, X_t) \}$$

(5.6)

Finally, each $X_{Ti}$ is labeled with the label of the closest manifold. Like the previous algorithms, its output is a mapping model.

### 5.6 Experimental Evaluation: Signal-level Algorithms

We experimentally assessed the efficacy of the signal level data mapping models using the same two datasets and data variation scenarios as was used in the case study in Chapter 4. We implemented all of the algorithms we developed and simulated parts of the proposed framework in Python. In the remainder of this section, we describe the experimental setup, and present results on different aspects, including runtime.
and accuracy.

5.6.1 Experimental Setup

Data Preparation. The data we used for both datasets is 3-dimensional accelerometer data. A sliding window of two second was used for data segmentation. Furthermore, interpolation techniques were applied on the data whenever the source and target domain time series sampling frequencies are different. For such purposes, we used the Spline algorithm implemented in the SciPy package of Python[75], [76]. We also, for the second dataset, generated inter-model variation by downsampling the signal-level data. For the REALDISP dataset, we used the accelerometer data for the sensor located on right calf. We also generate data for lower sampling frequency (i.e. 25Hz and 12.5Hz) scenarios by using available functions on SciPy (i.e. decimate) [75].

In sample selection (see the submodule named ”sample selection” in Fig. 7.2), only data segments with variance more than 0.8 were chosen so that standstill data segments are removed from the similarity search process. For example, we didn’t use the data associated with the activity 'Standing' in the first dataset, and the data associated with 'Frontal hand claps' in the second dataset. Therefore, we use 4 activities for the first dataset and 14 activities from the second dataset. In the reconfiguration phase we work with signal level data, and for activity recognition, in real-time execution, we transform the data into feature space.

Similarity Network-based Clustering. We first need to find the distance thresh-
old for each source domain to build the similarity network. We use a greedy approach to find the best distance threshold: 1) We initialize \( d_t = d_{avg}/2 \), where \( d_{avg} \) is the average distance in the entire similarity network; 2) We perform community detection on the similarity network with the current \( d_t \); 3) If the number of communities is lower than the number of activities, we decrease \( d_t \) by 0.02 and go back to step 2. Otherwise, we return the communities as clusters. Furthermore, we remove the small clusters (less than 2% of data) in the community detection phase. These small clusters are generally outliers (e.g. are wrongly labeled or are results of incorrect data collection procedure), and is therefore better to remove them before the knowledge transfer phase.

For community detection, we used the multi-level algorithm [77] because of its low time complexity. We used the implementation of the algorithm available in iGraph [78]. In each community, we choose the top-5 highest degree nodes as representative for the CADM algorithm. These nodes are taken as ”summarization” for their neighbors.

Motif Discovery. As mentioned in Section 5.4.4, we used a brute force motif discovery method [73] to build the dictionary \( DB_M \) for the MADM algorithm. We discovered top-\( k \) motifs for each activity. However, with each motif discovered, all of its occurrences are removed from the data. The minimum distance \( d_{th} \) that we used to find occurrences was 0.05. The method uses Euclidean distance to measure the
similarity between two data segments.

Mapping Model. In all three of the data mapping algorithms BFADM, CADM and MADM, we found the set of signal segment pairs by each algorithm (see the last line in each of Algorithms 2–4). We can use the segment pairs in two different modes: 1) to build a function to map the target data to the source data (i.e. regression model). The regression model uses the segment pairs to build functional models for each axis; and 2) to perform label mapping by assigning labels from the source domain data segments to the corresponding target domain data segment and re-train the computational model specifically for the target domain.

We used three different approaches as described below.

- **1-axis regression model**: for each axis of accelerometer in the mapped data, the regression model is built based on the corresponding axis in the target domain. We build this for the purpose of visualization. Using the 1-axis regression model, we visualize the applicability of functional data mapping for each variation category.

- **3-axes regression model**: for each axis of accelerometer in the mapped data, the regression model is built based on all three axes in the target domain. This model is more accurate because it also captures sensor mis-orientations compared to 1-axis regression model.

- **Label mapping**: for each multi-dimensional data segment in the target domain,
we assigned the label of closest data segment in the source domain. The newly labeled data is later used to re-train the computational model. We used this mapping to retrain the classification algorithms in the activity recognition.

5.6.2 Intermediary Results

Network-based Clustering Results. Table 5.1 shows the results obtained when the network-based clustering is applied on both datasets. The number of vertices in the similarity network and the number of clusters obtained are shown in the second and third columns of the table. The next three columns show the quality of the clusters obtained as measured by, minimum, average and maximum distance between vertices for all clusters. The last column shows the average distance between vertices in the entire (unclustered) network. The average cosine distance observed in the clustered networks show that the data points in each cluster are highly similar compared to the average distance observed in the entire similarity network. Also, the number of clusters obtained was found to be close to the number of activities, indicating that the clusters are meaningful.

Motif Finding Results. For each domain (e.g. device, model, placement, subject), we discovered 5 to 10 motifs from each activity. Using the discovered motifs, the size of the dictionary was reduced by up to two orders of magnitude. For each source
Table 5.1: Results on network-based clustering for both datasets. The three columns Min dist, Avg dist and Max dist show the minimum, average, and maximum cosine distance observed in each cluster. The last column shows the average distance in the entire network.

<table>
<thead>
<tr>
<th></th>
<th># Vertices</th>
<th># Clusters</th>
<th>Min dist</th>
<th>Avg. dist</th>
<th>Max dist</th>
<th>Avg. net dist (d_{avg})</th>
</tr>
</thead>
<tbody>
<tr>
<td>LG Nexus (Dataset 1)</td>
<td>2506</td>
<td>6</td>
<td>0.00</td>
<td>0.04</td>
<td>0.22</td>
<td>0.12</td>
</tr>
<tr>
<td>Galaxy S+ (Dataset 1)</td>
<td>3848</td>
<td>7</td>
<td>0.00</td>
<td>0.03</td>
<td>0.19</td>
<td>0.11</td>
</tr>
<tr>
<td>Ideal-Placement (Dataset 2)</td>
<td>1814</td>
<td>14</td>
<td>0.01</td>
<td>0.07</td>
<td>0.31</td>
<td>0.26</td>
</tr>
</tbody>
</table>

domain, the size of the dictionary is at most |N| \times 20 (|N| is the number of activities) time series which is significantly smaller compared to the case where all time series are stored in the dictionary.

5.6.3 Runtime Comparison

Fig. 5.2 shows the normalized runtime of the three data mapping algorithms BFADM, CADM and MADM for the case where the source domain and the target domain are the two Samsung S3 mini smartphones in Heterogeneity AR dataset. As we can see in the first dataset, for real-time execution, the MADM algorithm is the fastest since it has the smallest dictionary size among the three algorithms. On the other hand, the off-line processing time is the highest for MADM, as we need to run motif discovery to construct the dictionary of the source domain data. The reported times are for
Table 5.2: Comparison in terms of R-squared value for various data mapping algorithm/regression model combinations. The last column, SyncDM, is a baseline. The LG Nexus smartphones are used as the target and the source domains (Heterogeneity AR dataset).

<table>
<thead>
<tr>
<th></th>
<th>BFADM</th>
<th>CADM</th>
<th>MADM</th>
<th>SyncDM (baseline)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>X</td>
</tr>
<tr>
<td>1-axis reg. model</td>
<td>0.68</td>
<td>0.71</td>
<td>0.68</td>
<td>0.63</td>
</tr>
<tr>
<td>3-axes reg. model</td>
<td>0.73</td>
<td>0.76</td>
<td>0.71</td>
<td>0.73</td>
</tr>
</tbody>
</table>

90% overlap to segment the source domain data in the search process.

5.6.4 Data Mapping Accuracy

R-squared analysis. As mentioned in Section 5.6.1, we used two different regression models to build the data mapping functions. Table 5.2 shows the R-squared value of the two regression models (1-axis, 3-axis) for the three mapping algorithms BFADM, CADM and MADM. In all of the data mapping algorithms, we set the similarity threshold $\tau = 0.8$ for all three dimensions.

The last column in Table 5.2, SyncDM, is included to serve as a baseline. It is based on human activity data we collected for ten minutes using two different devices (a shimmer sensor and a Samsung Galaxy S4) at the same time. The data from these two devices were synchronized. From Table 5.2, it can be seen that the R-squared
Figure 5.2: Normalized runtime comparison for the three algorithms BFADM, CADM and MADM for both real-time and off-line data processing for the Samsung S3 mini smartphone from the Heterogeneity AR dataset. Since off-line data processing (in the reconfiguration state) typically happens only once, whereas real-time execution happens many times, we conclude that the algorithm MADM is by far the fastest algorithm of the three approaches.

Closer look at accuracy. Further, Fig. 5.3 shows scatter plots of the regression model (1-axis) of data mapping using the MADM algorithm for inter-subject, inter-
device, inter-model, and inter-placement knowledge transfer. In each case, we used linear regression with a polynomial of degree 5 to approximate the nonlinear behavior of the accelerometer data. We used the Scikit learning package to implement the regression models [79]. From Fig. 5.3, it can be seen that finding a relation between different subjects or placements using a linear regression model can not be captured.
Figure 5.4: Scatter plots and corresponding linear regression models for the data mapping algorithm MADM for inter-model and inter-placement heterogeneity on the 3 axes of accelerometer data.

(a) Inter-Model mapping.  
(b) Displacement mapping.

accurately compared to inter-device and inter-model scenarios. On the other hand, a linear regression model is found to be an effective model to mitigate device and frequency related variations. In all four scenarios, the variation of data along the linear regression model is due to the noisy data of the accelerometer. In the reconfiguration of computational model, we used the label mapping instead of direct use of data.
Choice of Similarity threshold $\tau$. Fig. 5.5 shows how the choice of the value for the similarity threshold $\tau$ affects the number of data segment pairs with the same label for both datasets. In these experiments, for the inter-subject scenario, we randomly chose a subject as target domain and the rest of the subjects as source domain for both datasets. For the inter-device scenario, we used two settings with different sampling frequency (LG Nexus and Samsung S3 for the first dataset, and original and downsampled data for the second dataset) as the target domain and the source domain respectively. For the inter-model heterogeneity scenario, we used the LG Nexus smartphones. For the inter-placement scenario, we use, for subject 8, the
self placement as the target domain and the ideal-placement as the source domain. All of the reported values in Fig. 5.5 are for the BFADM algorithm.

In general, we see that higher value for similarity threshold results in lower number of data segment pairs (plots against the left vertical axis (shown in red)). To gain insight into the potential tradeoff here, using the ground truth label of the target domain data, we computed the accuracy of one-to-one data segment mapping (data labeling). The corresponding results are plotted against the right vertical axis (shown in blue). It can be seen that, as the similarity threshold is increased, the number of pairs obtained decreases while the one-to-one data mapping accuracy increases. The best trade-off value for the similarity threshold depends on the heterogeneity type, but is generally observed to be around 0.7 for both datasets.

5.6.5 HAR Accuracy: Revisiting The Case Study

We return to the two datasets in the motivating case study discussed in Chapter 4 to evaluate the performance of the proposed framework. We built HAR models using different types of classifiers, including decision tree (DT), random forest (RF), and k-nearest neighbor (k-NN) algorithms just as we did in Chapter 4. We considered the Upper-bound as the baseline for our comparison (Chapter 4). We considered four scenarios as described below to evaluate the performance of the data mapping built based on the various algorithms.
**Figure 5.6:** Comparison of activity recognition accuracy with and without using transfer learning algorithms (Heterogeneity AR Dataset).

**Inter-Subject:** the computational model is built based on all the subjects except one which is used for testing. We only used one device setting (e.g. 100Hz sampling frequency). Therefore, the source domain is data from all subjects except one, and the target domain is the excluded subject. We used both datasets for this scenario.
(a) Inter-subject. 
(b) Inter-model. 
(c) Inter-placement.

Figure 5.7: Comparison of activity recognition accuracy with and without using transfer learning algorithms (REALDISP dataset).

The reported values are the average for all subjects.

Inter-Device: here we consider the Samsung Galaxy S3 data, and use one of the instances for training and the other for testing. Therefore, the source domain is data from the one device, and the target domain is the other device from the same model.
We only used the first dataset for this scenario.

**Inter-Model:** here we build the classifiers using the data of one specific sampling frequency and test it on a different model. Hence, the source domain is data of a specific model, and target domain is the data of other models. We used the Samsung S3 mini and LG Nexus smartphone for the first dataset. We also used 50Hz and 12.5Hz data from the second dataset.

**Inter-Placement:** here we build the classifiers using the ideal placement of all subjects except one as the source domain. We used the self-placement of the excluded subject as the target domain. We used the REALDISP dataset for this scenario. The reported values are the average for all subjects.

Fig. 5.6 shows the effect of knowledge transfer on the accuracy of activity recognition for the *Heterogeneity AR* dataset. We note the maximum increase in accuracy achieved due to transfer learning in the three variation scenarios, inter-subject, inter-device and inter-model. Those numbers are 6%, 13% and 15%, respectively. Fig. 5.7 shows the accuracy of activity recognition for the *REALDISP* dataset. We have an increase in the accuracy of activity recognition for all data mapping algorithms in general. The accuracy increases by up to 17%, 9% and 32% for inter-subject, inter-device and inter-placement scenarios, respectively. Random forest and decision tree classifiers show more accuracy boost when signal-level transfer learning is applied.
5.7 Experimental Evaluation: Manifold-based Algorithm

To evaluate our manifold-based, we used a publicly available human activity recognition dataset. The dataset we used here is the "Daily and Sports Activities Dataset" [68] (Section 4.2.5). We implemented all of the algorithms we developed in Python. We focused on inter-subject variation in the rest of this section. We only used periodic activities in our analysis. In the remainder of this section, we present results on the manifold learning-based algorithm.

5.7.1 Manifold Representation

Fig. 5.8 shows three different activity manifolds along with mapped data from the target subject. Each point corresponds to a micro-activity in the low-dimension manifold space. Fig. 5.8a and Fig. 5.8b show examples for the case where the manifold and the mapped data from the target subject are for similar activities. It is clear that the data point in the target domain are matched with the manifolds. On the other hand, Fig. 5.8c shows an example for the case where the manifold and the mapped data from the target subject are for different activities. It can be clearly seen that the mapped data is different compared to the learned manifold.
Figure 5.8: The data similarity for manifolds in the source and the target domain. The source and the target domain are different subjects (star: projected data, circle: learned manifold).

5.7.2 Labeling Accuracy

In this part of the analysis, we evaluate the accuracy of data labeling for the new subject data (i.e. target). We compare our manifold based transfer learning approach (MLTL) with two other methods: 1) Clustering-based labeling using K-MEANS algo-
Figure 5.9: The accuracy of data labeling.

Figure 5.10: The accuracy of activity recognition with/without MLTL.

Algorithm; and 2) Using training data for the same subject, a case that can be considered as upper bound on the accuracy (Upper Bound). Fig. 5.9 shows that the average accuracy of MLTL is 43% higher compared to clustering-based method, and it is only 6% lower compared to the upper bound.
5.7.3 **HAR Accuracy**

We built a random forest (RF) classifier using the new labeled data. Fig. 5.10 shows the increase in the accuracy when the data are labeled using the MLTL algorithm. Fig. 5.10 shows that, if we take one subject out, the accuracy drops by up to 41% for decision tree (DT) and by up to 27% for RF for the case when there is no knowledge transfer. In the worst case scenario, if we randomly use one subject to train the HAR model, the accuracy decreases by up to 61% for DT classifier.

5.8 **Summary**

We proposed a novel design framework to support autonomous reconfiguration of wearable systems under different kinds of data variations. In the framework, context and configuration change in the system is monitored in the real-time execution state, and when sufficient change is detected, a procedure is activated in the reconfiguration state to autonomously rebuild the computational model. As part of the framework, we developed four data mapping algorithms for extracting knowledge in the reconfiguration state. These algorithms are designed combining effective methods for signal similarity identification, motif discovery, clustering, and manifold learning. Focusing on activity recognition as a case study, we performed analysis using 3-axis accelerometer data in three different datasets. We also experimentally evaluated the performance of our framework and data mapping algorithms. From the experimental
evaluation, we found that our framework yields an accuracy increase ranging between 3% to 40% compared to the case where a reconfiguration phase is absent from the system.
CHAPTER 6. INTERACTIVE RECONFIGURATION:

ACTIVE LEARNING

In this chapter, we develop algorithms for interactive reconfiguration of mobile health monitoring systems. We focus on settings where the learner can gain knowledge from sensory system (other sensors and smart environment) and human (physicians, users) experts in the reconfiguration phase.

6.1 Introduction

Active learning is an effective paradigm for a solution toward interactive reconfiguration of M-health monitoring systems. The key idea in active learning is to first select informative instance(s) from the set of unlabeled data to label, and then re-run the learning algorithm to minimize prediction and decision making errors [35, 80, 81]. However, current active learning approaches rely on unrealistic assumptions, especially for domains such as mobile health monitoring. In real M-health applications, it is possible (and likely) to have multiple sources of knowledge with different levels of confidence (certainty) or areas of expertise. Here, we call each source of knowledge an expert (e.g. user, physician, another sensor, smart environment). Considering uncertainty of experts, a multi-expert system that implements collaboration between experts can result in lower overall cost and more accurate prediction model(s). In-
deed, with the emergence of IoT platforms, cooperation between sensory devices is becoming increasingly feasible and is likely to be a crucial aspect of future M-health systems [82]. We target a system that strives to minimize the cost of active learning from experts while maintaining the system’s accuracy. The cost associated with an expert may be defined in terms of user inconvenience, expert’s feedback cost, equipment’s cost, power consumption, etc.

We outline our specific contributions below along with a road map of this chapter. We propose a novel architecture designed for multi-expert M-health systems. The framework manages different aspects the interactive management unit as well it’s communication with other units. The key idea is to keep the system’s uncertainty below a pre-specified threshold while minimizing the overall cost of re-training the learning algorithm. Fig. 6.1 gives a high level overview of the architecture. We formulate the multi-expert active learning problem as a joint optimization of which data instances to select and which experts to use to annotate the informative samples. The objective is to minimize the active learning cost subject to a pre-defined uncertainty threshold. The formulation is presented in Section 6.3.

Furthermore, we have developed two algorithms corresponding to two different aspects of the architecture. The first algorithm utilizes the training data from the most similar context in order to initialize the learner of the system. The second algorithm manages the source(s) of knowledge for each query in the active learning
process. The algorithm is designed to handle different types of experts. All of the algorithms we have developed are presented in Section 6.4. We evaluate the proposed architecture using two different human physical activity datasets as case studies. The first dataset is on daily activities (e.g. walking, sitting), and the second dataset is on workout activities (e.g. running on a treadmill). We show that using the architecture increases accuracy of activity recognition by up to 45% for the daily activities and by up to 35% for sport activities compared to the case where the architecture is not used. We also show that the number of queries from costly experts is reduced by 78%. Details of the experimental setup we employed are presented in Section 6.5, and the experimental results are presented in Section 6.6.

6.2 System Architecture

A conventional mobile monitoring system is generally composed of several sensor nodes, a base station, and a back-end server. Each sensor node is attached to the body (a wearable or an implant device) to sample and preprocess physiological signals and transmit partial results (e.g. features) to the gateway. Moreover, sometimes additional sensors are placed in the environment (e.g. smarthome) to capture the context as well as the decision making process. The gateway is a more powerful unit, such as a smartphone, that performs data fusion for the health analytic system. The results are further transmitted, typically through the Internet, to a back-end server for storage, further processing, and clinical decision support from physicians.
Figure 6.1: High-level illustration of the cost-optimal multi-expert active learning (Co-MEAL) architecture for M-health monitoring systems.

and health-care experts.

With this description of conventional architecture of M-health systems as a background, our proposed reconfigurable architecture is constituted as depicted in Fig. 6.1. It consists of three main components: 1) databases (DB), 2) a data processing unit (DPU), and 3) an expert management unit (EMU). The DPU is the core intelligence of the M-health system. The DPU may be located entirely on the gateway or on both the gateway and the back-end server depending on the computational power of the gateway. The DB, the EMU, and the experts can be considered part of the back-end server.

The DPU communicates with the DB and the EMU whenever there is a need for
system reconfiguration. The incoming data is sampled by the DPU, and then the DPU decides whether or not to request labels. If the learner is capable of classifying the data accurately using the labeled data in the current context, the active learning step is deactivated. The DPU has multiple submodules each of which is responsible for a different functionality in the reconfiguration phase. We discuss in the remainder of this section the details of each submodule, the Databases, and the EMU.

6.2.1 Data Processing Unit: Learner Initialization

In active learning, a learner often begins with a small number of instances in the labeled training set $L$. However, an M-health system quite often may have to start with no labeled data at all. This requires more sophisticated strategies in the initialization phase of the active learning process. Meanwhile, for many applications of M-health, there may be data for related contexts. For example, in a physical activity monitoring system, the data from other subjects could be used to initialize the classifier of the system. In Section 6.4, we present an initialization algorithm which uses the related labeled data to build the first learner model.

6.2.2 Data Processing Unit: Query Strategy

In order to reduce annotation cost, we need a query strategy to select the best subset of unlabeled instances. The decision of whether or not to query an instance can be framed in several ways (refer back to Section 2.3). In the framework, we use a hybrid
query strategy of the above approaches that is based on a random forest classifier.

The informativeness of an input instance is computed as the mean informativeness of
the trees in the forest. The informativeness measure of a single tree is the fraction of
samples of the same class in a leaf.

6.2.3 Data Processing Unit: Databases

A key component in our active learning architecture is a set of databases. Each
database represents knowledge for a unique case. Below, we formally define a database.

Definition 7 (Database). A database (or dictionary) $DB$ is an unordered set of
instances. The $i^{th}$ instance is a $q$-dimensional vector denoted by $DB_i$, where $q$ is
number of features extracted from each instance. Each instance may or may not be
labeled in the dictionary.

In our architecture, there are three types of dictionaries. The target unlabeled data
dictionary stores the incoming streaming data in real-time. This data will be used
both to initialize the system and to modify the classifier. The second database, target
labeled data, gradually collects two kinds of information: (i) instances from the third
database, related labeled data, and (ii) queries from the EMU. The database referred
to as related labeled data is built based on related contexts and domains useful for
initializing the classifier and decreasing the cost of queries.
6.2.4 The Expert Management Unit

In real-world health-care applications, labels may come from various sources, including different health-care experts, other mobile devices or even a smart environment. Furthermore, the knowledge provided by each expert is subject to uncertainty. For example, the performance of human annotators depends on factors such as level of expertise, experience, and concentration/distraction [36]. Additionally, experts may have more or less knowledge for different parts of the problem that has to be solved (e.g., different classes, different features). Another factor is the cost of data annotation by each expert. Therefore, it is necessary to design a system that is capable of interacting with multiple experts efficiently.

The multi-expert architecture we envision satisfies several unique properties. 1) The learning process relies on the collective intelligence of a group of experts. 2) The architecture could also provide a learning opportunity for experts. In particular, our system allows for each expert to update their knowledge based on the feedback from other experts in each round of query. 3) The expert selection submodule chooses the expert(s) in such a way that the query cost is minimized, while keeping the provided label for the queried instance accurate. 4) The architecture leverages different points of view from the experts to provide a more comprehensive model through the system’s life time. For example, new class labels may arise during the life time, or a new sensor may be integrated into the mobile health system. 5) An expert can be part of the
data processing unit. For example, in a multi-node mobile system, one node shares the label with the DPU whenever the node is confident. We present the algorithm we developed for expert selection in Section 6.4.

6.3 Problem Statement

Let $L = \{\ell_1, \ell_2, \ldots, \ell_k\}$ be $k$ groups of data that the system needs to classify (label). Let $E = \{e_1, e_2, \ldots, e_p\}$ be a set of $p$ possible experts each of which has full or partial knowledge of the current labeled set $L$. We model the knowledge quality of each expert on label set $L$ using an uncertainty score vector. For a given expert, its uncertainty score vector $\vec{U} = (u_1, u_2, \ldots, u_k)$ is a vector whose $i^{th}$ entry stores a value in the range $[0, 100]$ (i.e. percentage) indicating the uncertainty of the expert on its prediction of the label $\ell_i$. A number close to 0 indicates high expertise. Our learner is considered as one of the experts in the sense that it has its own uncertainty score vector.

We are now ready to formally define the collaborative multi-expert active learning problem:

**Problem 2** (Co-MEAL Problem). Let $X_U = \{x_1, x_2, \ldots, x_n\}$ be the sequence of data instances in the new context or configuration, the labels of which are not yet known with certainty. Let $X_L = \{(x_{n+1}, y_{n+1}), (x_{n+2}, y_{n+2}), \ldots, (x_{n+m}, y_{n+m})\}$ be the set of labeled data. This labeled data may come from related labeled data $X_R$ or partially
trained learner (i.e. prior knowledge). The Co-MEAL problem is to select a subset \( X_S \subset X_U \) to ask from the set of heterogeneous experts \( E \) such that the overall data annotation cost is minimized and uncertainty of the learner model \( M_{lr} \) trained on the set \( X_S \cup X_L \) is lower than a pre-specified threshold in the new context.

We formulate the problem as a joint optimization of which data instance(s) to select and which expert to use to annotate. The objective is to minimize the active learning cost under the pre-defined uncertainty threshold:

\[
\text{Minimize } C_{al}(M_{lr}) \text{ subject to } \vec{U}_{M_{lr}} \leq \vec{U}_{th} \tag{6.1}
\]

where \( \vec{U}_{th} \) is the acceptable (threshold) amount of uncertainty in the system’s performance, \( M_{lr} \) is the learner model, \( \vec{U}_{lr} \) is the uncertainty of the learner, and \( C_{al}(M_{lr}) \) is the cost of query of the selected instances from the set of experts.

Equation (6.1) can be rewritten by incorporating query strategy and expert selection into the objective function. Let \( \beta_i \) be a binary variable that determines whether or not sample \( x_i \) is selected as an informative instance:

\[
\beta_i = \begin{cases} 
1, & \text{if label for } x_i \text{ is needed} \\
0, & \text{otherwise.} 
\end{cases} \tag{6.2}
\]

Let \( \alpha_{ij} \) be a binary variable indicating whether or not the expert selection model requests the label for instance \( x_i \) from expert \( e_j \). That is:
\[ \alpha_{ij} = \begin{cases} 
1, & \text{if label for } x_i \text{ is asked from } e_j \\
0, & \text{otherwise} 
\end{cases} \quad (6.3) \]

And let \( c_{ij} \) be the query cost for instance \( x_i \) from expert \( e_j \) and \( p \) be the number of experts. Then, the Integer Linear Programming (ILP) formulation for the Co-MEAL Problem is as follows:

\[
\text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{p} \beta_i \alpha_{ij} c_{ij} 
\]

subject to:

\[
\sum_{j=1}^{p} \alpha_{ij} = 1 \quad \forall \ i \in \{i \mid \beta_i = 1\} \quad (6.5)
\]

\[
\beta_i \in \{0, 1\}, \quad \alpha_{ij} \in \{0, 1\} \quad (6.6)
\]

\[
\vec{U}_{Mr} \leq \vec{U}_{th} \quad (6.7)
\]

The constraint (6.5) guarantees that each selected unlabeled data is asked exactly from one expert and the constraint in (6.6) ensures that the variable \( \beta_i \) and \( \alpha_{ij} \) take only binary values.

It is extremely difficult to optimize the Co-MEAL Problem directly, since even the simplified version of the problem is NP-hard. The simplified problem is the case where
the selected samples from the related data are known and we only need to find the
optimal set of experts to annotate the unlabeled selected instances. The minimization
is over the entire set of potential sampling sequences and experts with different costs,
an exponentially large number. Furthermore, the learner $M_{lr}$ is updated with each
additional example and we can only calculate this effect after we know which examples
are chosen and labeled, which makes it impossible to directly solve the problem.

6.4 Algorithms

We instead developed greedy algorithms to find effective, approximate solution for
the Co-MEAL Problem. We use available query strategies to find the most informa-
tive data instances as we discussed in Section 6.2.2. In this section, we present the
algorithms we developed for learner initialization and expert selection as well as the
driver algorithm for the overall functionality of the architecture.

6.4.1 Driver Algorithm: Collaborative Active Learning

The driver routine for solving the Co-MEAL problem is presented in Algorithm 9.
The algorithm takes as input the dictionary containing other contexts’ data $DB_R$, the
dictionary of target labeled data $X_L$ (which could be an empty set or could contain
a small number of instances at first), the dictionary of target unlabeled data $X_U$,
the label set $L$, the number of instances for each round of query $K$, the uncertainty
threshold vector $\vec{U}_{th}$, and the expert set $E$. It delivers as output the model $M_{lr}$ which
Algorithm 6 CollaborativeActiveLearning-Driver

Input: $DB_R, X_L, X_U, L = \{\ell_1, \ell_2, \ldots, \ell_k\}, K, \bar{U}_{th}, E$

Output: Learner $M_{lr}$

1: $DB_L \leftarrow \text{InitializelLearner}(X_U, DB_R, L)$
2: for all $x_i \in X_U$ do
3: if old knowledge is applicable then
4: $X_L \leftarrow X_L \cup (x_i, y_i), X_U \leftarrow X_U \setminus (x_i, y_i)$
5: end if
6: end for
7: $X_L \leftarrow X_L \cup DB_L, X_S \leftarrow \emptyset$
8: $M_{lr} \leftarrow \text{Classifier}(X_L)$
9: while $\bar{U}_M > \bar{U}_{th}$ do
10: $X \leftarrow \text{getInformativeInstances}(K, X_U)$
11: for all $x_i \in X$ do
12: $y_i \leftarrow \text{Annotate}(x_i, E, M_{lr}(x_i), \bar{U}_{th})$
13: $X_S \leftarrow X_S \cup (x_i, y_i)$
14: end for
15: $\text{broadcastLabeledData}(X_S)$ /*collaborative*/
16: $\text{updateUncertaintyAll}(X_S, E)$ /*collaborative*/
17: $X_L \leftarrow X_L \cup X_S, X_U \leftarrow X_U \setminus X_S, X_S \leftarrow \emptyset$
18: $M_{lr} \leftarrow \text{UpdateLearner}(X_L)$
19: end while

is retrained with the augmented labeled data.

Algorithm 9 calls the routine InitializelLearner (Algorithm 7) to determine the closest data instances in the related contexts $DB_R$. It then expands the target labeled database $X_L$ with them. In the next step, the architecture uses the current learner model to label the target domain whenever prior knowledge is applicable (lines 2–5). This is useful whenever the data distribution is the same but we need to upgrade the system’s configuration (e.g. new sensors, new set of labeled data).
The while-loop represents the data annotation phase. There are two steps in each iteration of the while-loop: 1) the most informative instances are chosen via the function GETINFORMATIVEINSTANCES (see the query strategy submodule in Section 6.2.2), which returns the $K$ most informative instances from the database of target unlabeled data; and 2) the routine ANNOTATE (Algorithm 8) obtains labels from confident-enough, cost-efficient experts for each selected instance.

Then, the new labeled data instances augment the target labeled database and re-train the learner model $M_{lr}$. The routine BROADCASTLABELEDDATA conveys the new labeled data to the other experts. The UPDATEUNCERTAINTYALL routine is run on each expert to re-calculate their uncertainty score based on the new gathered informations (i.e. new labeled instances). This process is repeated until the learner’s uncertainty scores falls below the uncertainty threshold $U_{th}$ for all $\ell_i \in L$. The learner re-trained based on the enhanced target labeled data database is returned as the output of the driver algorithm for the Co-MEAL architecture.

6.4.2 Learner Initialization: Transfer Learning

As was mentioned earlier, we need to use the available knowledge when data distribution in the source domain(s) is different from that in the target domain (i.e. current context). The INITIALIZEL Earner routine (Algorithm 2) gets as input the database of labeled data for related contexts $DB_R$, the database of unlabeled data for the current context $X_U$, and the label set $L$ including $k$ types of labels.
Algorithm 7 INITIALIZELearner

Input: $DB_R, X_U, L = \{\ell_1, \ell_2, \ldots, \ell_k\}$

Output: $DB_L$

1: $DB_L \leftarrow \emptyset, SIM \leftarrow \emptyset$
2: $(Centers_u, Clusters_u) = CLUSTERING(X_U, k)$
3: for all $DB_R_i \in DB_R$ do
4:   $SIM_i = \text{cosSim}(Centers_{DB_R_i}, Centers_u)$
5:   $SIM \leftarrow SIM \cup SIM_i$
6: end for
7: for all $\ell_i \in L$ do
8:   $DB_{\ell_i} \leftarrow \text{argmax}_\ell SIM$
9:   $DB_L \leftarrow DB_L \cup DB_{\ell_i}$
10: end for

The database of related contexts $DB_R$ may include none to several source(s) domains (e.g. different sensor locations, different subjects). The algorithm finds $k$ clusters for the unlabeled data (line 2). It then searches the entire database of related data $DB_R$ to find the closest set of labeled data to each cluster, and returns these data as the output. Note that labeled data for each $\ell_i \in L$ may come from different source domains. We employed cosine similarity to measure the closeness of clusters in the current contest and labeled data in the new context [26].

6.4.3 Expert Management: Active Learning

One focus of this chapter is the heterogeneity (e.g. uncertainty, type, and cost) of experts. Our system should be cost-sensitive and we need to minimize the cost of active learning. In real-world scenarios for mobile health monitoring, the system can actively communicate with other sensory systems, the user (through the interface),
the physician (through the back-end server), etc. In addition to type of an expert, one other major problem in health monitoring is the subjectivity of the expert’s decisions. For example, the performance of human annotators depends on factors such as level of expertise, experience, and concentration/distraction [36].

The first task in our expert management in active learning context is to categorize different types of experts and then find a cost-sensitive solution for the problem. We categorize the experts into three different types as follows (see also Figure 6.2).

- **Perfect Expert**: A perfect expert supplies correct labels for every query, but the cost of data labeling is high. Our goal is to minimize the number of queries from such experts. An experienced physician, a multi-sensor smart health facility center, a supervised data collection system are all examples of perfect experts.

- **Imperfect Expert**: An imperfect expert provides labels for each query, but the validity of a label is questionable. In other words, the expert is knowledgeable in only part of the data distribution and it may provide incorrect labels. Examples
of imperfect experts are inexperienced doctors, the user, and sensory systems with limitations (e.g. limited number of sensors). The cost of labeling by an imperfect expert is lower, but still we need to minimize the number of queries from such experts.

- **On-demand Expert:** An on-demand expert, in contrast to the other types of experts, provides labels with little or zero cost. However, their knowledge is limited (e.g. a subset of label set $L$) and they may not be able to reply to every query. Other wearable sensors with different configurations are examples of on-demand experts.

The system should be capable of approximating the uncertainty of experts over the label set $L$. This is possible because the reputation and/or performance of experts can be estimated based on available reviews. For example a wristband activity tracker may be known to be an expert on detection of walking with different paces, or an experienced doctor may be well-known for her/his expertise. Based on the above categorization, we propose a greedy algorithm for expert selection that minimizes the cost of data annotation (Algorithm 8).

The algorithm `ANNOmATE` takes as input the instance $x$, the set of experts $E$, a semi-label $L$, and the uncertainty threshold $\bar{U}_{th}$. The semi-label $L$ is the most probable label(s) for instance $x$ based on the learner’s current knowledge. The goal of the algorithm is to find the least costly and most confident expert for the instance $x$. The


Algorithm 8 Annotate

Input: $x, E$, semi-label $\mathcal{L}, U_{th}$
Output: label $\ell$

1: $u = 100$
2: for all on-demand expert $e_i \in E$ do
3: \hspace{1em} $\mathcal{L} \leftarrow \text{MajorityVote}(x)$
4: \hspace{1em} $u \leftarrow \text{MinUncertainty}(e_i, \mathcal{L})$ in labeling $x$
5: end for
6: while $u > U_{th}(\mathcal{L})$ do
7: \hspace{1em} $E_{conf} \leftarrow \emptyset$, $C_{conf} \leftarrow \emptyset$
8: \hspace{1em} for all perfect or imperfect expert $e_i \in E$ do
9: \hspace{2em} if $U_i(\mathcal{L}) \leq u_{th}$ then
10: \hspace{2.5em} add $e_i$ to the $E_{conf}$ /*confident experts*/
11: \hspace{2.5em} add $\text{cost}(e_i)$ to $C_{conf}$ /*cost of experts*/
12: \hspace{2em} end if
13: \hspace{1em} end for
14: \hspace{1em} expert $\leftarrow \text{argmin}_{e_i \in E_{conf}} C_{conf}$
15: \hspace{1em} $\mathcal{L} \leftarrow \text{label from expert}$ for instance $x$
16: \hspace{1em} $u \leftarrow \text{uncertainty of expert}$ in labeling $x$
17: end while
18: $\ell = \mathcal{L}$

expert selection method is performed in two phases. First, the expert management unit asks the on-demand experts to provide labels. The semi-label $\mathcal{L}$ and its related uncertainty change if on-demand experts are more confident about another specific label based on majority voting. If the uncertainty for the new semi-label $\mathcal{L}$ is still below the threshold, the second phase is performed to ask from the other experts (i.e. imperfect and perfect experts).

The set $E_{conf}$ is the set of experts that are confident enough to detect the semi-label $\mathcal{L}$ at a given moment during the course of the algorithm. The semi-label $\mathcal{L}$ is
updated in the course of the algorithm, and is deemed acceptable if the uncertainty score of the selected expert is lower than the threshold $\tilde{U}_{th}(\mathcal{L})$. Further details are shown in Algorithm 8.

6.4.4 Variations on the Driver Algorithm

Before concluding the current section on algorithms, we include a brief discussion on a matter that is related to our experiments. Specifically, in our experiments, we consider three different variation of Algorithm 1 in order to show the effects of different properties:

- CAL corresponds to the normal operation of the Co-MEAL architecture with both transfer learning and collaboration, but with no prior knowledge available. This is a variant of Algorithm 9 in which lines 2–5 are excluded.

- CAL* corresponds to the case where learner(s) have been in the system with less knowledge and the active learning phase is used to upgrade the system with changes such as adding a new sensor or new labels. CAL* is variant of Algorithm 9 in which the lines 2–5 are included.

- NCAL corresponds to the operation of the Co-MEAL architecture with transfer learning and without collaboration. NCAL is simply a variant of Algorithm 9 in which the lines 15 and 16 corresponding to collaboration between experts are excluded.
The way data is collected for CAL* requires a bit of clarification. Whenever the subject uses the system with the new configuration, those samples for which the system is confident to label are added to the target labeled database \((X_L)\).

6.5 Experimental Setting

Human activity recognition (HAR) is an important component of M-health systems. For example, it enables novel context-aware monitoring in elderly care and health care for patients with chronic diseases. It also provides standalone applications, for example, for measuring energy expenditure. Various types of sensors have been utilized in mobile devices, including gyroscopes, magnetometers, accelerometers, GPS sensors, or combinations of different sensor modalities [83, 55, 56, 84]. We therefore chose activity recognition as our case study. We consider two datasets with two distinct focuses: real-world dataset and sport and daily activity dataset. In rest of this section, we describe the design setting we employ to simulate real-world scenarios.

6.5.1 Data Processing

A moving average filter is applied on the activity data to remove high frequency noises from inertial sensors (i.e. preprocessing). For the first dataset, after the data filtering, a two-second sliding window was applied for data segmentation, and we consider 50% overlap between consecutive windows. The second dataset is segmented into windows of five seconds. For each data segment, a variety of features can be
extracted from each segment of data. We extracted statistical features that have been pointed out in previous studies as being effective for activity recognition [70, 71]. More details on the statistical feature set used in the experiments are presented in Table 4.1 (Section 4.3.2).

The next step in the data processing is classification. Previous research in the field of activity recognition has shown that decision tree, random forest (RF), K-nearest neighbors, and Support Vector Machine classifiers can be effective [84, 58]. In this chapter, we use a RF classifier for both the learner model and the sensor-based experts. We did so for two reasons. First, RF works as a committee of decision tree classifiers on different sub-samples of the data and uses averaging to refine the predictive accuracy to control over-fitting. Secondly, using RF, we are able to measure the uncertainty of machine learning models over the unlabeled target data (Section 6.2.2). We used the K-means clustering algorithm to categorize the unlabeled target data (CLUSTERING($X_U, k$) in Algorithm 2).

6.5.2 Simulation of Experts

As we described in Section 6.4.3, our architecture involves three different types of experts. Each expert has its own properties such as type, uncertainty, and cost. The cost of an expert is proportional to its confidence level over the label set $L$. We split the available labeled data into expert initialization and evaluation subsets, with a ratio of 50 percent for each subset for each location (e.g. head, shin). We use the
expert initialization to simulate the experts and the evaluation data to validate the proposed architecture. We assume that all of the devices are experts, except for the one which we assume is the learner. The various expert types are simulated as follows.

**Perfect Expert.** We used the ground truth label as the least uncertain (most confident) expert in the expert management unit. In a real-world scenario, this expert can be a highly confident physician or the user in the monitoring system. In order to be more realistic, we also allow for the possibility of error made by a perfect expert. We set the probability of error between 1% to 10% depending on the difficulty of detecting the specific label.

**Imperfect Expert.** We model an imperfect expert using the sensors from other locations (as opposed to the learner’s location). However, in order to generate imperfect experts, we partly use the expert initialization data. For example, we only use 5% of the expert initialization data for each label. If we need to decrease the uncertainty of an imperfect expert, we add more labeled data to build its classifier. We estimate the uncertainty score vector for each expert based on their prediction error on the evaluation data. We assume that, in real-world scenario, an expert is aware of its own knowledge level.

**On-demand Expert.** Each on-demand expert is trained in such a way that it is only confident over a subset of the label set $L_{selected} \in L$. To simulate each on-demand expert, we use all the expert initialization data for $\ell_i \in L_{selected}$ to train the classifier,
and a few instances for other label.

6.6 Experimental Validation

We implemented all of the algorithms we developed and the various simulations of the proposed architecture in Python. In this section we present experimental results. The presentation is organized in two parts. Section 6.6.1 presents results demonstrating the benefits of using active learning on physical activity monitoring, whereas the focus of Section 6.6.2 is on cost evaluation of the architecture.

6.6.1 Performance Evaluation Scenarios

To assess the performance of the Co-MEAL architecture, we consider three change scenarios (corresponding to the cases discussed in Section 4.1): context change (new subject), configuration change (new sensor) and user need change (new activity).

Context Change (New Subject). In this scenario, a wearable sensor is placed on human body as a learner (e.g. wristband activity tracker). We assume that we have labeled data for other subjects, however, the current subject’s activity patterns might be different compared to them. Therefore, the operating context of the system is different from what we have in the training labeled data. Our goal is to teach the sensor for the new context (i.e. the new subject).

We initialize the learner classification algorithm using Algorithm 7 for the current context. Then, the query strategy submodule finds the most informative set of
Figure 6.3: Comparison between the collaborative active learning algorithm (CAL) and two baseline algorithms: randomly selected samples in active learning (RAL) and a simple transfer learning based approach (STL). The scenario is context change.

The query strategy submodule selects the top $K = 10$ informative instances in each round of query. We compare the activity recognition accuracy of the collaborative active learning (CAL) algorithm with two baseline algorithms: 1) randomly select samples in the query procedure instead of informative samples (RAL) and 2) employ a simple transfer learning-based learner (STL) which adds the most similar training data of other subjects in each iteration. We repeated the above scenario for 8 subjects in the two datasets for all three algorithms. In each re-run of this scenario the excluded subject is the new context and the rest are considered as related
contexts.

The activity recognition accuracy results we obtained for this scenario are summarized in Fig. 6.3. The reported results are averages of the various re-runs. We consider the case where the sensor is located on the waist of user. The left plot in Fig. 6.3 shows results for dataset 1 (daily activities) and the right plot shows results for the dataset 2 (sport activities). The horizontal axis shows the percentage of queries for the CAL/RAL algorithms and the percentage of used training data for the STL algorithm.

Fig. 6.3 shows that by querying only 15% of data, the accuracy of CAL reaches 88% and 90% for dataset 1 and dataset 2. The RAL algorithm’s accuracy increases monotonously through the active learning process until a point where the added labeled data has no impact on the learner’s accuracy. This shows that the new labeled data offers no information gain if we don’t have an appropriate query strategy. The accuracy of the STL algorithm varies for most cases as we increase the amount of training data from other subjects. After a point, the difference in distribution of training data and our context result in an uncertain learner for the STL algorithm.

Configuration Change (Sensor Addition). Here we consider a configuration change scenario in which new sensors are added to the system for better performance. To simulate the scenario, we looked at a learner with two sensors located on different body parts. Each node has a prior knowledge over a subset of label set $L$ with high
Recalling the descriptions of CAL* and CAL given in Section 6.4.4, we compare the activity recognition accuracy for (1) CAL, (2) CAL*, and (3) using training data for the same configuration, a case that can be considered as upper bound on the accuracy (Upper Bound). We compare these three algorithms for the scenario just described. Fig. 6.4a and Fig. 6.4b show that adding a new sensor with prior knowledge over the label set $L$ increases the activity recognition accuracy by up to 25% for the daily activities and by up to 48% for the sport activities.

**User Need Change (New activity).** The scenario considered here is a user need change in which a user adds new activity (new data to classify). To simulate this,
(a) Add two new labels (daily activities)  (b) Add two new labels (sport activities)

**Figure 6.5:** Performance of the collaborative active learning algorithm under a user need change scenario.

we again look at a learner with two sensors located on different body parts (as in the previous paragraph), but in this case we let the sensory system require to label two new groups of data (i.e. activity type). Fig. 6.5a and Fig. 6.5b) show that for this case in which the sensors needed to learn new classes of data, the initial accuracy of CAL* increases by up to 40% compared to CAL for both datasets.

### 6.6.2 Query Cost Evaluation

In this section, we present results assessing the effects of uncertainty threshold for imperfect experts, number of on-demand experts, and benefits of collaboration on cost of active learning in the Co-MEAL architecture.
Figure 6.6: Expert uncertainty threshold versus activity recognition accuracy and number of queries (left axes: number of queries, right axes: accuracy).

Uncertainty Threshold, Imperfect Experts. Depending on importance of accurate data classification in our system, we can control the cost of queries from perfect and imperfect experts in the EMU unit. In other-words, we can trust experts with higher level of uncertainty in the data annotation process because of the higher uncertainty threshold that the system can tolerate. We run the EMU unit for various range of uncertainty thresholds. We only consider perfect and imperfect experts to remove the role of on-demand experts from the overall cost of data annotation. Fig. 6.6 presents the effects of uncertainty threshold on (i) the number of queries from human annotators and (ii) the activity recognition accuracy. Note that the plots in Fig. 6.6 use two vertical axes to show the two quantities simultaneously.
Figure 6.7: total number of queries (active training data) versus activity recognition accuracy and number of human queries (left axes: human queries, right axes: accuracy).

In these experiments, we assumed that the most expensive data annotator is a human, and see how our expert selection can shift the queries from the human expert to imperfect experts if we alter the uncertainty threshold. The lower the uncertainty score threshold is, the higher the number of queries from human(s). However, even with higher uncertainty threshold, we still reach an acceptable activity recognition accuracy. We have reported results for both datasets. We only employ 20% of data to annotate in both datasets. For $\bar{U}_{th} = 0.2$, the accuracy of activity recognition increases by up to 90% for the daily activities and by up to 95% for sport activities. The number of queries from human(s) reduces by up to 32%.

**Number of On-Demand Experts.** The cost of active learning varies depending
on the type of experts. In this subsection, we consider another real-world case where there are multiple on-demand experts in the system. As mentioned in Section 6.4.3, an on-demand expert provides label data with little or zero cost in the active learning phase. Each on-demand expert is trained in such a way that it has knowledge in one fourth of $\ell_i \in L$. Fig. 6.7 shows the accuracy of the system for the case that we have an expensive expert and several on-demand experts (4 and 6 for daily and sport datasets). As we can see, as we move forward in the active learning process, the number of expensive queries remains below 15% and 5% for daily and sport activities datasets, while the accuracy increases steadily.

Fig. 6.8 shows how number of on-demand experts changes the number of queries from expensive experts. The higher the number of on-demand experts, the lower the number of queries. Fig. 6.8a and Fig. 6.8b show the number of queries from the human for different number of on-demand experts. Higher number of on-demand experts covers a larger part of the data distribution, and consequently the number of queries from the human expert(s) decreases. The total number of queries is 10% for both datasets. The results are shown for different subjects.

**Benefits of Collaborative Active Learning.** We simulate a real world scenario with all three types of experts. We set the cost for each expert proportional to their confidence on the label set $L$. The set of experts is sorted in a descending order based on their cost and we named them based on this order (E1 has the highest
Number of On-demand Experts

(a) Dataset 1 (daily activities)

(b) Dataset 2 (sport activities)

Figure 6.8: Number of on-demand experts versus number of queries from human expert(s).

cost). Fig. 6.9 presents the total number of queries for both collaborative (CAL) and non-collaborative (NCAL) algorithms. The goal of this experiment is to show how the collaboration between expert set reduces the cost, and increase the activity recognition accuracy.

The training data size corresponds to the number of queries from the set of experts. As we query more data instances from the set of experts, the queries from a human expert for the CAL algorithm decreases at a faster rate compared to the non-collaborative version (i.e. NCAL). Furthermore, confidence of on-demand and imperfect experts increases in CAL and the number of queries from them increase, and consequently the cost of data annotation is reduced.
Figure 6.9: Comparison of non-collaborative and collaborative active learning in terms of total number queries. Each color corresponds to a different expert. The number of queries from imperfect (E2, E3) and perfect (E1) experts decreases in the collaborative learning, because on-demand experts ((a) E4-E7, (b) E4 & E6) increase their knowledge in the previous rounds.

6.7 Summary

We explored the use of active learning and transfer learning for reconfiguration of mobile health monitoring systems. Firstly, we proposed an expert selection algorithm which can be used in different real-world scenarios with heterogeneous set of experts. The expert selection module is incorporated as a core part of the expert management unit within the Co-MEAL architecture. The expert management unit cooperates with the query strategy module and the transfer learning (i.e. learner initialization) module to further decrease the cost of the active learning process. Secondly, the
Co-MEAL architecture is designed to allow collaboration among experts in order to reduce annotation cost and enhance knowledge of inexpensive experts for future data labeling.

We demonstrated the efficacy of the expert selection algorithm within the Co-MEAL architecture using activity recognition as a case study. We designed several real-world scenarios for two different human activity datasets for cases where a system faces configuration, context, or user need change. We showed that high accuracy (over 90%) can be attained by annotating only a small fraction (about 10%) of unlabeled data. We also showed that cost-optimal expert selection and collaboration among experts enables big savings in annotation cost.
CHAPTER 7. ADAPTABLE REPRESENTATION
LEARNING: ACTIVE DEEP LEARNING

In this chapter, we propose a deep learning architecture targeted for human activity recognition (HAR) systems with two main purposes. We proposed a hybrid deep architecture which can learn presentations from related data and personalize the model for the current situation of the system.

7.1 Introduction

Current machine learning algorithms rely on engineered features to address increasingly complex activity recognition systems. Human activity recognition is challenging due to the large variability and heterogeneity in human activity data [55, 85, 57]. We need to learn a better representation of data to build generalized models. Learned representations often result in much better performance than can be obtained with hand-designed representations [53, 38, 60, 86]. Secondly, a system is designed for a limited number of predefined protocols (e.g. a fixed sensor location, a fixed set of activities) or operating environments (e.g. a specific group of subjects). Therefore, the accuracy of machine learning algorithms drops whenever the system’s setting or operating environment changes [87, 52]. Thirdly, the collected data also includes non-relevant activities and non-activities in addition to the activity of interest through
the life time of the system, which make it necessary to have a specific representation for each activity. The HAR system should be able to interact with the users to update the activity recognition algorithms. However, these ways may cause disturbance or interference in user’s life and may incur expenses related to active learning [22, 23, 26, 34, 88].

One step further, we revisit the model selection and interactive reconfiguration units from Chapter 1 and propose an active deep learning architecture in order to provide robust activity recognition for HAR. This architecture has the potential to uncover features that are tied to the dynamics of human motion production, from simple motion encoding in lower layers to more complex motion dynamics in upper layers. The complexity of a deep neural network can help mitigate the effects of variations in the data such as walking with different paces. Furthermore, the architecture also capture the relation between consecutive data segments. Finally, we already have a built-in uncertainty measurement to chose informative instances for active reconfiguration of HAR model.

The remainder of this chapter is organized as follows. We provide a formal statement of the underlying robust HAR problem in Section 7.2, and in Section 7.3, we present the active deep learning architecture. Later, in Section 7.4, we illustrate the details of deep network layers. Finally, in Section 7.5, using two different activity recognition datasets, we evaluate the system’s performance.
7.2 Problem Statement

In real-world scenarios, it is impossible to collect training data for every context of an HAR system. There is a need to develop systems which are capable of contextually modifying an existing machine learning algorithm. A related labeled database $X_R$ may be used to initiate the machine learning algorithm. However, the initialized model may be inaccurate due to difference between the related data distribution and the current context.

Whenever a user starts using an HAR system, context-specific unlabeled data $X_U$ is collected gradually throughout the system’s life time and may well be different from the related labeled data. The goal is that the learner successfully recognizes activity types with an accuracy up to an upper bound (i.e. having training data in the current context) by solving these two subproblems: 1) The system comprehends any changes in the distribution of activity data; and 2) The system is designed with the capability of finding the most informative set of unlabeled instances for the purpose of context-awareness.

Our goal is to minimize the cost of annotation from the set of experts, while getting close to the upper bound accuracy. An expert can be another sensor, a smart environment, or the user [34]. We can decrease the annotation cost in two ways: 1) Have a better initialized model using the related labeled data; or/and 2) Design an optimal query strategy to select the best set of instances in the current context data.
We provide a formal definition of cost-optimal active learning as follows.

Let $A = \{A_1, A_2, \ldots, A_k\}$ be a set of $k$ activities that the system aims to recognize. Suppose each instance of data is a 3D tensor, corresponding to different dimension of different sensors. Let $X_R = \{(x_{r_1}, y_{r_1}), (x_{r_2}, y_{r_2}), \ldots, (x_{r_m}, y_{r_m})\}$ be the set of labeled related data which are used to initialize the learner. Using the $X_R$ data, we can initialize the learner model in $2^m$ ways $M = \{M_1, M_2, \ldots, M_{2^m}\}$.

Let $X_U = \{x_{u_{m+1}}, x_{u_{m+2}}, \ldots, x_{u_{m+n}}\}$ be the sequence of observations made by wearable devices (e.g. smartphone, wristwatch), the labels of which are not yet known with certainty. We formally define the minimum cost active learning (MinCAL) problem as follows.

**Problem 3 (MinCAL).** The MinCAL problem is to select a subset $X_S \subset X_U$ to query from the expert set, and select a model $M_i \subset M$ such that the overall cost of query is minimized and the overall accuracy $\Lambda$ of the learner model for activity recognition is greater than a threshold $\lambda$. The accuracy threshold $\lambda$ is always considered lower than the upper bound accuracy for the available sensory system.

The MinCAL problem can be formulated as an optimization problem as follows. Let $\alpha_i$ be a binary variable that determines whether or not the query strategy submodule selects $x_{u_i}$ as an informative instance, and whether or not the selected model is trained using $x_{r_i}$ ($x_i \in X_R \cup X_U$):
\[ \alpha_i = \begin{cases} 
1, & \text{if label for } x_i \text{ is selected} \\
0, & \text{otherwise}. 
\end{cases} \tag{7.1} \]

And let \( c_i \) be the query cost for an instance from the set of experts. We assume that the cost for all \( x_{ri} \subset X_R \) is zero in the formulation. However, the way in which we select the \( x_{ri} \in X_R \) indirectly effects the informativeness of unlabeled instances. Thus, the Integer Linear Programming (ILP) formulation for the MinCAL problem is as follows:

\[ \text{Minimize} \quad \sum_{i=1}^{m+n} \alpha_i c_i \tag{7.2} \]

subject to:

\[ \Lambda \geq \lambda \tag{7.3} \]

\[ \alpha_i \in \{0, 1\}, \; \forall \; i, j \tag{7.4} \]

The objective function in (7.2) aims to minimize the total cost of the queries from the set of experts by both choosing the best initial learner model and best set of informative instances. (7.3) ensures that the variable \( \alpha_i \) take only binary values. The constraint (7.4) guarantees that the accuracy of the final activity recognition is more than the threshold accuracy.
The problem above is an integer programming problem, which in general is known to be NP-Hard. Rather than solving this problem exactly, we propose a deep learning architecture which greedily tries to minimize the number of queries. The proposed architecture exploits deep learning to deduce a generalized model for the initialization. This model can be used with any active learning approach to further minimize the number of queries.

### 7.3 Closed-Loop Deep Activity Recognition

The proposed active deep architecture, to be used for interactive reconfiguration of wearable HAR, is depicted in Fig. 7.1. The reconfiguration is performed in two phases. The first phase aims to pre-train the deep neural network using the related labeled data. The second phase is used to actively update the deep neural network by getting feedback from experts to annotate informative unlabeled data. We will go through details of these two steps and related system development matters in the rest of this section.

**Pre-training Phase: Offline.** Supervised learning forms the core intelligence of the activity recognition system, which analyzes the training data to map raw data into activities. The term *offline* here is used to signify that the pre-training is performed on the back-end server prior to real-time execution of activity recognition. Using the related labeled data $X_R$, as shown in Fig. 7.2, the deep neural network is trained for activity recognition. Our deep network model is fed with activity data segments
Figure 7.1: High level overview of the proposed active deep learning architecture.

through all layers. The network is trained to: 1) capture the structure of the data \((representation\ learning)\); and 2) act as the first activity recognition system \((context-related)\). The \(context-related\) layers are trained for the related context and depending on the similarity between the related data and the current context, a subset or all of the layers are considered as \(representation\ learning\) layers. This pre-trained model will be used for both the real-time execution and the active learning phase.

**Active Learning Phase: Real-time.** In the real-time execution, each sensor node is attached to the body (a wearable or an implant device) to sample and preprocess physiological signals and transmit partial results (e.g. features) to the gateway. The gateway is a more powerful unit, such as a smartphone, that performs data fusion for the HAR. The system gradually collects the unlabeled data \(X_U\). Whenever the active learning phase is enabled (e.g. operating in a new context), the most informative set of unlabeled data are selected to fine-tune the weights in the \(context-related\) layers.
Algorithm 9. The corresponding algorithm for the active deep architecture is presented in Algorithm 9 in pseudocode form. The algorithm takes as input the related labeled data $X_R$, the set of unlabeled data $X_U$ in the new context, the activity set $A$, the number of instances for each round of query $K$, and the uncertainty threshold $u_{th}$. It gives as output the deep neural network model for HAR $M_{HAR}$.

Algorithm 9 calls a routine $\text{InitializeDNN}$, which corresponds to the pre-training phase, using the related labeled data $X_R$. The while-loop subsequent to the network initialization corresponds to the active learning phase. In each iteration of the while-loop, the most informative instances are selected via the function $\text{TOPK}$ (recall the
discussion about information density in Section 2.3), which returns the $K$ instances with the highest information density from the unlabeled data set $X_U$.

Then, the new labeled data are added to the labeled data and the network parameters are updated using the new labeled data (UpdateDNN). This process is repeated until we iterate over all the selected instances or the learner’s uncertainty score falls below the threshold $u_{th}$ for all $\mathcal{A}_i \in A$. The $u_{th}$ corresponds to the threshold accuracy $\lambda$ that we defined in the problem formulation—lower $u_{th}$ corresponds to higher $\lambda$. In practice, it is easier to measure the uncertainty of the model rather than the accuracy on the unlabeled data instances. The model $M_{HAR}$ is returned as the output of the algorithm. In the next section, we discuss the details of the deep architecture that we propose.

### 7.4 Deep Learning Architecture

The proposed deep learning architecture is depicted in Fig. 7.2. The network comprises eight layers with weights—five Convolutional layers, two recurrent LSTM layers, and one soft-max layer. The deep network model works by passing each activity data segment through the CNN layers to produce fixed-length vector representations for the next layer. In the output of each CNN layer, we have different levels of abstraction for the activity data. After feature learning, the LSTM layers then take over to capture the temporal dependence of activity data.
Figure 7.2: An illustration of the 3D-ConvLSTM architecture. The network’s input is 3-dimensional. The network includes 11 layers including the max-pooling layers.

The output of the last LSTM layer is fed to a soft-max layer which produces a distribution over the activity labels. Max-pooling layers, with size of $2 \times 1$, follow the first, fourth, and fifth Convolutional layers. The ReLU and hyperbolic tangent non-linearity is applied to the output of Convolutional layers and LSTM layers, respectively.

The first Convolutional layer filters the 3D input tensor with 128 kernels of size $5 \times 1 \times 3$ with a stride of 1 sample. The second Convolutional layer takes as input the (response-normalized and pooled) output of the first Convolutional layer and filters it with 96 kernels of $5 \times 1 \times 128$ size. The third Convolutional layer has 64 kernels of size $5 \times 1 \times 96$ connected to the outputs of the second Convolutional layer. The fourth Convolutional layer has 48 kernels of size $5 \times 1 \times 64$, and the fifth Convolutional
Figure 7.3: The 3D input structure with the sliding window representation for data segmentation.

layer has 32 kernels of size $5 \times 1 \times 48$. Because we use zero padding to compute the convolution, the second dimension of the data fed through the network remain the same unless we have max-pooling (e.g. first layer).

The output of the fifth Convolutional layer is reshaped to flatten the second and third dimensions of data. This flattened input is then fed into the recurrent layers of the network. Both recurrent layers feature 128 LSTM units. This is then fed into the fully connected layer, where the soft-max activation function is applied in order to compute the most likely class that the action falls within. Below, we describe specific features that we exploit in the deep learning architecture.

3D Input Structure. The input to the model is considered as a 3-dimensional time
series sensor data of the form, $len_w \times num_s \times 3$, where $len_w$ represents the window length and $num_s$ represents the number of sensors. Inspired by RGB representation of images, we add a notion of depth, corresponding to the X, Y and Z components of each sensor module (e.g. accelerometer, gyroscope). The input structure is illustrated in Fig. 7.3.

**Max-Pooling.** Max pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change. Also, the number of parameters decreases because of max-pooling layers and therefore, the training time decreases.

**Built-in Active Learning.** The soft-max layer produces a distribution over the activity labels. Using the output of the soft-max layer, we compute the base informativeness $\phi^{SE}$ (Section 2.3) for all the collected unlabeled instances $x_{ui} \in X_U$. This is used to calculate the information density in the active learning phase.

**Implementation.** We implemented the proposed deep architecture in the Keras library. The pre-training phase and active learning phase are run on a Tesla K80 GPU. We used a fully-supervised model training using back-propagation. The network parameters are optimized by minimizing the cross-entropy loss function using mini-batch gradient descent. The pre-training and the active learning phases are performed in the back-end server, and the trained deep architecture is returned to the gateway.
Table 7.1: Detailed specifications of the used datasets.

<table>
<thead>
<tr>
<th></th>
<th># of locations</th>
<th># of 3D sensors</th>
<th># of activities</th>
<th>Type</th>
<th>Sampling Freq.</th>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opportunity</td>
<td>17</td>
<td>37</td>
<td>17 (+ Null)</td>
<td>Complex</td>
<td>30Hz</td>
<td>Gyro, Acc, Mag</td>
</tr>
<tr>
<td>Real World</td>
<td>7</td>
<td>21</td>
<td>8</td>
<td>Simple</td>
<td>50Hz</td>
<td>Gyro, Acc, Mag</td>
</tr>
</tbody>
</table>

for real-time execution.

7.5 Experimental Evaluation

In this section, we experimentally assess the efficacy of the proposed architecture using real human activity data. We consider two datasets with two distinct focuses. The first dataset relates to complex daily activities (referring back to Section 4.2.3), while the second dataset focuses on simple body movements and postures (referring back to Section 4.2.4). A comparison of these two datasets is presented in Table 7.1. The section is organized as follows. We present the architecture design settings in Section 7.5.1. In Section 7.5.2, we evaluate the performance of the deep architecture with/without active learning. In Section 7.5.3, we compare this work to state-of-the-art deep HAR (DeepConvLSTM [42]).

7.5.1 Architecture Design Settings

In this subsection, we go over the design setting we used in the data processing flow.

Data Preparation. We applied a moving average filter on the activity data to remove high frequency noises from the sensor data. After data filtering, a two-second
sliding window was used for data segmentation. Therefore, dimensions of input tensors are $60 \times 37 \times 3$ and $100 \times 21 \times 3$ for Opportunity and Real World datasets, respectively. We consider the 50\% overlap (one-second) between consecutive segments of activity data.

**Deep Network Setting.** We train the deep network (3D-ConvLSTM) with batches of 100 data segments. We used the RPM Prop optimizer, in which the learning rate is 0.001 and the decay factor is 0.9. We also shuffle the batches at each training epoch for the pre-training phase. We applied dropout as a regularization technique with $p = 0.5$ probability during the pre-training phase. Because the number of instances that are queried to annotate are scarce, we revoke the dropout setup in the active learning phase. For the DeepConvLSTM system, we used the same setting as authors mentioned in their paper [42].

**Feature Extraction.** A variety of features can be extracted from human activity data segments. Previous studies have shown that statistical features such as mean, standard deviation, peak to peak, maximum, and minimum are effective for activity recognition systems [55, 34]. We used the features listed in Table 4.1 to generate feature space data for the traditional classifiers.

**Active Learning Parameters.** The query strategy submodule selects the top $K = 20$ instances with the highest information density in each query round to ask from the annotators. We use the ground truth labels to label the selected samples. We set
\( \beta = 1 \) for the information density calculation.

### 7.5.2 Performance Evaluation

We compare our 3D-ConvLSTM architecture with three traditional classifiers using engineered features: decision tree (DT), random forest (RF), and K-nearest neighbor (K-\textit{nn}) algorithms. We also compare our architecture with the state-of-the-art architecture—DeepConvLSTM [42]. For the purpose of careful comparison, we define several scenarios as follows.

**Fully-Trained.** The training and test set have the same data distribution (e.g. same subject, fixed sensor location). This scenario gives an upper bound on classification accuracy. In other words, we have models which are fully-trained for the current context (fixed-locations, and same train and test subjects).

**Subject-Independent.** The computational model is built based on all subjects except one, which is used for test. This scenario is used to evaluate the effect of subject heterogeneity on the performance of the HAR models. This scenario shows how different computational models (i.e. traditional classifiers, deep architectures) behave in the presence of variation.

**Active-Training.** In this scenario, each machine learning algorithm is initially built as a \textit{subject-independent} model. We actively update the model by adding labeled data using the active learning submodule in order to increase the accuracy of the HAR system. We use this scenario to show how the active learning approach increases the
accuracy of all of the machine learning models, including traditional classifiers and deep architectures.

To compare performance, we use the $F_1$-score for all models. The $F_1$-score combines precision and recall accuracy. The rationale for using $F_1$ score is that activity data classes are imbalanced and hence using accuracy may not be a good way to
compare their performance. Formally, $F_1$-score is defined as follows:

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$  \hspace{1cm} (7.5)$$

Fig. 7.4 shows bars corresponding to $F_1$-scores of HAR for different classifiers for the subject-independent scenarios. Subject 1 corresponds to the case that the machine learning models are trained on all subjects except subject 1. Because the Null class in the Opportunity dataset constitute approximately 75% of the data, we build the HAR models for both cases—with and without Null class.

Fig. 7.4a shows the $F_1$-score for all machine learning algorithms trained on the Opportunity dataset excluding the Null class. The deep networks are pre-trained for 100 epochs. We can see that our proposed deep architecture enables an increase in $F_1$-score by up to 14% and 40% compared to DeepConvLSTM [42] and traditional classifiers.

Fig. 7.4b shows the $F_1$-score for the case where Null class is included. The average $F_1$-score of the 3D-ConvLSTM architecture is higher by 13%, 8%, and 11% compared to DeepConvLSTM, RF, and Knn algorithms. Fig. 7.4c presents the $F_1$-score for the RealWorld dataset for several subjects. Although the other classifiers performs as well as the deep architectures for some of the subjects, the average $F_1$-score is higher for both deep architectures. It can be seen that our architecture consistently gives $F_1$-score higher than 80%, while the $F_1$-score for other classifiers is fluctuating when
the test subject is changed. This shows that the 3D-ConvLSTM architecture is robust against subject heterogeneity. \( DT \) classifier shows the minimum \( F_1 \)-score for both datasets.

Fig. 7.5 and Fig. 7.6 show \( F_1 \)-scores for machine learning models equipped with the active learning submodule for the \emph{Opportunity} and \emph{RealWorld} datasets, respectively. We compare our architectures with: 1) a random forest based classifier; and 2) DeepConvLSTM HAR architecture [42]. We choose RF classifier because its \( F_1 \)-score for \textit{subject-independent} model is higher compared to the other classifiers as was shown in Fig. 7.4.

Fig. 7.5 shows that by using only 20\% query, the proposed deep architecture outperforms DeepConvLSTM and RF classifier by up to 12\% to 25\%, respectively. Furthermore, the \( F_1 \)-score for our proposed architecture scales up with a higher slope compared to the other models. The \( F_1 \)-score is only 1\% to 3.2\% less than the upper bound accuracy for a \textit{fully-trained} deep architecture. The number of epochs is set to 100 in the active learning phase for the \emph{Opportunity} dataset.

Fig. 7.6 shows the \( F_1 \)-score values for the \emph{RealWorld} dataset. Although sometime DeepConvLSTM and RF start with higher \( F_1 \)-score values, the proposed architecture outperforms them by less than 10\% query. For the \emph{RealWorld} dataset, the final \( F_1 \)-score is only 1\% under the \( F_1 \)-score of a \textit{fully-trained} model in the worst case scenario.
Figure 7.5: F1-score for different subjects in the active learning phase versus the number of queries for the Opportunity dataset. 400 instances roughly are 20% of data for each subject.

(i.e subject 11). We see that the final accuracy goes up to 100% for several subjects. The number of epochs is set to 50 in the active learning phase for the Real World dataset. We run each experiment 10 times and present the average in all of the above experiments.
Figure 7.6: F1-score for different subjects in the active learning phase versus the number of queries for the Real World dataset. 400 instances roughly are 15% of data for each subject.

ConvLSTM architecture, the F1-score increases monotonously in most cases for both datasets. This shows that the features learned by our architecture are more robust to variation and noise in the activity data.
7.5.3 Deep Architecture Results

Number of Epochs. The number of epochs determines the time that we need to pre-train or train the final deep neural network. We explore the effects of the number of epochs on the $F_1$-score of the deep architectures. Fig. 7.7 shows the $F_1$-score of our architecture and DeepConvLSTM [42] architecture for different number of epochs. The reported accuracy are the average for 10 different runs for all subjects when tested on the Opportunity dataset. As we increase the number of epochs, we observe a raise in the $F_1$-score for both architecture. Furthermore, our architecture shows better results as we increase the number of epochs compared to DeepConvLSTM [42].

The accuracy follows the same trend for the Real-World dataset. We observe that the deep networks need fewer epochs for the Real-World dataset because of the simpler set of activities. We reach to maximum accuracy after 26, 40, and 100 epochs for Real-World, Opportunity with Null, and Opportunity without Null datasets, respectively.

Execution Time Comparison. In this subsection, we compare the runtime of training for our deep architecture (i.e. 3D-ConvLSTM) with the DeepConvLSTM architecture. We train both networks for all subject-independent scenarios using the Opportunity dataset (i.e. four deep networks). The results are summarized in Table 7.2. We ran the training phase for all scenarios for 10 epochs, and report the average of runtime. The training time for the 3D-ConvLSTM architecture is one or-
Table 7.2: Comparison of training time of different subjects for each epoch (3D-ConvLSTM vs DeepConvLSTM).

<table>
<thead>
<tr>
<th>Deep Architecture</th>
<th>Time per epoch (second)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S1</td>
</tr>
<tr>
<td>3D-ConvLSTM (this work)</td>
<td>8.2</td>
</tr>
<tr>
<td>DeepConvLSTM ([42])</td>
<td>56</td>
</tr>
</tbody>
</table>

Figure 7.7: The accuracy of activity recognition vs the number of epochs in the training phase (comparison between 3D-ConvLSTM and DeepConvLSTM).

Generally, we need to run the training phase for 50 to 100 epochs. Therefore, our architecture is three orders of magnitude faster compared to the state of the art architecture (i.e., DeepConvLSTM). The max-pooling layers and fewer number of kernels decreases the number of parameters in the deep architecture, which in turn decreases the training time for the 3D-ConvLSTM architecture.
**Figure 7.8:** The average accuracy for all subjects and the percentage of number of parameters that are needed to set versus the number of frozen layers in the active learning phase. The number of queries is 20% of data for the test subject. The total number of parameters in the deep network is 921473.

**Number of Frozen Layers.** Here we investigate the quality of learned representation in the pre-training phase. As we mentioned earlier in Section 7.3, based on the similarity of related data and the current context, we may retrain one or multiple *representation learning layers*. Fig. 7.8 shows the effect of number of frozen layers on the $F_1$-score and the number of parameters that we need to optimize in the active training phase. Fig. 7.8 shows the slight changes observed when we change the number of frozen layers.

These result show that the learned features are representative of the data, and they can be used in a related context (i.e. another subject). Furthermore, as the number of frozen layers increases, the number of parameters that needs to be fine-
tuned decreases. This results in less time to update the pre-trained deep network in
the active learning phase. Fig. 7.8 shows that if we freeze all Convolutional layers,
the number of parameters to modify decreases by 13% in the active learning phase.

7.6 Summary

Human activity recognition plays a central role in health-care, fitness, and sport
because of its potential to enable novel context-aware data processing. With the rising
preponderance of wearable devices, there has been a large influx in the availability
of human activity data. Analysis and interpretation of the heterogeneous and high-
volume streaming data associated with HAR holds immense potential for impact.
We proposed a novel deep architecture to support reconfiguration of wearable HAR
systems capable of handling high volume of time-varying data. The architecture
includes an active learning module; whenever the system faces uncertainty (i.e. the
ID score for incoming samples is high) in the real-time execution, the active learning
phase is triggered to update the deep architecture.

The effectiveness of many traditional methods relies on domain knowledge and
feature engineering. Here, we used a deep architecture to learn hierarchical representa-
tion of features from raw sensor data as well as to capture the time dependencies of
human activity data. We also add an active learning submodule to the system whenever
it is necessary to get feedback from users for better human activity recognition.
This allows us to choose the best samples for fine-tuning the model to the new setting
that the system operates in (e.g. a new user). We demonstrated the efficacy of the algorithms as well as the architecture using real data of human activity collected in different contexts.
CHAPTER 8. ENERGY-EFFICIENT RECONFIGURATION MANAGEMENT

In this chapter, we present an energy-efficient hierarchical reconfiguration management framework that uses computationally simple screening units for detection of data variation scenarios. The goal is to detect changes in the setting of the system, while the power consumption is minimized.

8.1 Introduction

When used in real world scenarios, wearables face different types of data variations, such as mis-orientation, displacement, and misplacement of sensors, which dramatically impact performance. For example, one can train a classifier, based on a training dataset, to perform activity recognition. The classification algorithm, however, fails to detect physical activities, if the location of the on-body sensors changes [55]. The reconfiguration management unit (RMU), coupled with real-time execution, tracks anomalies and changes in the status of the system. This unit should initiate reconfiguration whenever it detects any variation in context, configuration, and/or user need. Depending on the case, it is crucial to take corresponding actions to mitigate the effect of such variations on the performance of the machine learning algorithm(s). From
now on, we use *data variation* and *anomaly* interchangeably referring to scenarios driving reconfiguration.

In the current paradigm for M-health systems, researchers make several unrealistic assumptions in the design of algorithms for reconfiguration management: 1) the RMU is only responsible for a specific kind of data variation [89, 90, 91]. For example, in an activity recognition system, the focus is only on displacement of sensor(s). As a result, these techniques require separate units for each data variation type; 2) although data variations happen rarely, occurring at most 10% of the times [92], current algorithms require constant execution of several anomaly management units (AMUs), an approach that demands extremely high computational power, given the stringent constraint resources of mobile devices; and 3) the RMU unit is application-specific and it is required to design RMU for each application separately. Fig. 8.1 shows the high level illustration for the conventional approach.

The aforementioned drawbacks with the current RMU approaches warrant development of new frameworks that not only are accurate in detecting anomalies and mitigating the effect of such events on the performance but also are energy-efficient. Inspired by power management techniques in design of digital systems, we propose a framework with the following properties: (1) our framework uses computationally simple screening units for detection of sparse anomalies; (2) a novel approach is introduced to capture the collective contribution of various signal attributes (i.e., features)
Figure 8.1: Current reconfiguration management approaches in networked wearables.

This approach combines detection and mitigation of anomalies into one module called anomaly management unit (AMU).

in predicting existence of an anomaly in the network as well as the costs associated with the features; this method is used to develop simple yet effective Anomaly Screening Unit (ASU); (3) we introduce a prediction graph model to capture inter/intrasensor energy consumption related to each feature for anomaly detection; (4) we propose a greedy algorithm to solve the cost-effective feature selection problem using the prediction graph model; and (5) we validate the efficiency of our approach using real data collected with motion sensors for physical activity monitoring.

8.2 Proposed Framework

A unique attribute of our reconfiguration management framework, as shown in Fig. 8.2, is an Anomaly Screening Unit (ASU) which is used to detect different types of
anomaly. ASU composed of several screening units (SUs) which each one is used to detect one or several types of anomaly (e.g. misplacement, displacement).

Followed by the ASU is a set of mitigation units (MUs) which each one is responsible to address a specific types of anomaly. As soon as an anomaly is detected, the screening unit activate the corresponding MU(s) through activation unit. Because of sparsity of anomalies, decoupling anomaly screening and anomaly mitigation results in energy saving given that screening units are designed to be computationally inexpensive and communicate infrequently.

Furthermore, in our framework, management is independent of the application computational model (e.g., activity recognition, energy expenditure estimation, etc).
In other words, reconfiguration management is performed on the sensor data without considering the underlying application. The main benefit is that the ASU is transferable to different types of application with the same set of sensors.

ASU is the core component of our framework in delivering highly energy-efficient management of anomalies. The goal is to design SUs such that they are energy-efficient capable of remaining active all the time, but also accurately activate the MUs upon occurrence of an anomaly. Our approach to development of each screening unit (SU) is based on the concept of information redundancy. It involves selecting an optimal set of features (in a training phase) and using the selected features during real-time execution of the system.

A high-level illustration of the two phases is shown in Fig. 8.3. In real-time execution, a sensor node within the network acquires a set of redundant features for screening. These features may be retrieved from another node, upon request, or extracted locally. Clearly, it is more desirable to use local features to avoid data transmission between sensor nodes. The set of minimum-cost redundant features are used to construct a computationally simple classifier to determine anomaly occurrence. In Section 8.3.2, the problem formulation for building an individual screening unit in training phase is provided.
8.3 Anomaly Screening Unit

In this section, we present our cost-efficient feature selection approach to construct computation model of SUs to detect anomaly. This approach identifies a set of redundant features with minimum computation and communication cost. We first introduce a new metric, called prediction index, which uses the inter-feature correlation and estimates the potential power of two features for predicting an anomaly. For this part, it is needed to have training controlled data and data in presence of anomaly as shown in Fig. 8.3. The prediction index is then used to construct a graph model, called Prediction Graph, which is further employed to find a set of feature pairs in constructing SUs. For each anomaly type, we construct a specific prediction graph. Finally, a set of energy efficient features are selected from each prediction graph to build individual screening units which together construct the anomaly screening unit (ASU).

8.3.1 Energy Model

We define energy models for both conventional and our framework as shown in Fig. 8.1 and Fig. 8.2, respectively. The energy consumption of a mobile sensor node consists of three components: (1) data sampling; (2) feature extraction; and (3) machine learning algorithms. According to Fig. 8.1, the total energy consumption can be calculated by


text goes here

Figure 8.3: The process of developing the anomaly screening module.

\[ E(\text{conv}) = E(ml) + E(s) + \sum_{k=1}^{N} E(RMU_k) \]  

(8.1)

where \( E(\text{conv}) \) represents energy consumption of the conventional models, \( E(ml) \) represents energy consumption of the application algorithm (e.g., activity recognition), \( E(s) \) denotes energy consumption due to data sampling, and \( E(RMU_k) \) is energy consumption of the \( k \)-th anomaly management unit.

The total energy consumption of our uncertainty management framework, shown in Fig. 8.2, can be written as:

\[ E(\text{scr}) = E(ml) + E(s) + \sum_{k=1}^{N} \alpha_k \times E(MU_k) + \sum_{i=1}^{M} E(SU_i) \]  

(8.2)

where \( E(\text{scr}) \) denotes energy consumption of the proposed model based on anomaly screening, \( E(SU_i) \) represents energy consumption of the \( i \)-th screening unit, and \( \alpha_k \)
indicates the activation frequency for \(SU_k\) and is given by

\[
\alpha_k = (F_k \times tp_k + (1 - F_k) \times fp_k)
\]  

(8.3)

where \(tp_k\) and \(fp_k\) denote true positive rates and false positive rates of the \(SU_k\) classifier respectively, and \(F_k\) represents the frequency of occurrence of the \(k\)-th anomaly type. These rates indicates how often a mitigation unit \((MU_k)\) need to be activated by corresponding \(SU_k\). We note that, in this formulation, the energy consumption due to feature extraction is considered as a part of the processing energy consumption for RMU, SU and MU units.

8.3.2 Problem Definition

Let \(S = \{s_1, s_2, \ldots, s_n\}\) be a set of \('n'\) sensor nodes forming a networked wearable system. Furthermore, let \(F_i = \{f_{i1}, f_{i2}, \ldots, f_{im}\}\) be a set of \('m'\) features extracted from sensor node \(s_i\).

**Definition 8** (Prediction Index). *Given two features \(f_{ij}\) and \(f_{kl}\), prediction index is defined as*

\[
\lambda_{ij}^{kl} = \frac{U_R(f_{ij}, f_{kl})}{U_A(f_{ij}, f_{kl})}
\]  

(8.4)

where \(U_R\) denotes correlation between the two features during normal operation of the network while \(U_A\) represents inter-feature correlation associated with presence of an anomaly. Here, we use symmetric uncertainty for correlation measurement. Sym-
metric uncertainty captures non-linear correlation between variables and therefore, is a safe measure for feature analysis studies [93, 94].

**Definition 9** (Symmetric Uncertainty). The symmetric uncertainty between two discrete random variables $X$ and $Y$ is given by:

$$U(X,Y) = \frac{2 \times I(X,Y)}{H(X) + H(Y)} \quad (8.5)$$

where $H(X)$ and $H(Y)$ represent the entropy of random variables $X$ and $Y$, respectively, and $I(X,Y)$ denotes the information gain between the two variables. $I(X,Y)$ is defined by:

$$I(X,Y) = H(X) - H(X|Y) \quad (8.6)$$

Symmetric uncertainty is the normalized information gain and is always between 0 and 1, where $U=1$ means that knowing the value of either variable can completely predict the other variable, and $U=0$ indicates that the two variables are completely independent.

**Definition 10** (Strong Pair Features). Two features $f_{ij}$ and $f_{kl}$ are considered as a strong pair, if the prediction index associated with them exceeds a given threshold, $\Delta$:

$$\lambda_{ij}^{kl} \geq \Delta \quad (8.7)$$

**Definition 11** (Prediction Graph). The prediction graph $G = (V,E,C_V,W_E)$ is an undirected graph with the following specifications. The variable $V$ denotes the set of
$m \times n$ vertices associated with the features extracted from the sensor nodes in $S$; $E$ represents the set of edges such that an edge $e_{ij}^{kl}$ exists if features $f_{ij}$ and $f_{kl}$ form a strong pair as defined previously. The set $C_V$ represents vertex weights. A weight $c_{ij}$ is assigned to each vertex $v_{ij}$ to represent the cost of feature $f_{ij}$ computation. Furthermore, $W_E$ represents edge weights. Each edge $e_{ij}^{kl}$ has a cost $w_{ij}^{kl}$ given by

$$w_{ij}^{kl} = \begin{cases} 
\beta, & \text{if } i \neq k \\
0, & \text{otherwise}
\end{cases}$$

(8.8)

where $\beta$ is the cost of data transmission between two different sensor nodes $s_i$ and $s_k$.

Using the prediction graph, our goal is to build SUs as shown in Fig. 8.3. We define the problem that we need to solve as follows.

**Problem 4** (Min-Cost Redundant Feature Selection). **Min-Cost Redundant Feature Selection (MCRFS)** is to select a subset of features (i.e. graph vertices) such that the total cost in terms of computation and communication is minimized, while we have at least one feature from each sensor node.

Given that wearables are used in a highly dynamic environment, the measurements are likely to be noisy, which may reduce the effectiveness of feature pairs in predicting anomalies. In order to further enhance the robustness of the system, our proposed framework provides flexibility in reliability level of the feature selection approach, which selects different sets of features. Motivated by K-connectivity in wireless sensor networks, we define K-reliable feature selection as follows [95]:
Problem 5 (K-Reliable Feature Selection). Given an exhaustive set of features, K-reliable feature selection is defined as finding K set of features which meet the conditions of our framework. In each iteration, the selected edges are removed from prediction graph for the next round.

The MCRFS problem can be formulated as follows. Assume $a_{ij}^{kl}$ is a binary variable indicating whether or not the edge $e_{ij}^{kl}$ exists in the prediction graph $G$. That is:

$$a_{ij}^{kl} = \begin{cases} 
1, & \text{if } e_{ij}^{kl} \in E \\
0, & \text{otherwise}
\end{cases} \quad (8.9)$$

The goal is to choose a subset of the edges such that all sensor nodes have at least one feature candidate in the final feature set. By choosing an edge from the prediction graph, we can perform a cross-feature analysis to determine whether or not the system has encounter an anomaly. Let $x_{ij}^{kl}$ be a binary variable indicating whether or not the edge $e_{ij}^{kl}$ is selected by our feature selection algorithm. Thus,

$$x_{ij}^{kl} = \begin{cases} 
1, & \text{if } e_{ij}^{kl} \text{ is selected} \\
0, & \text{otherwise}
\end{cases} \quad (8.10)$$

Thus, the optimization problem can be written as follows.

$$Z = \sum_{i,k=1}^{n} \sum_{j,l=1}^{m} a_{ij}^{kl} (c_{ij} + w_{ij}^{kl} + c_{kl}) x_{ij}^{kl} \quad (8.11)$$
Minimize \( Z \) \hspace{1cm} (8.12)

Subject to: \( \sum_{k=1}^{n} \sum_{j,l=1}^{m} x_{kl}^{ij} \geq 1, \quad \forall i \in \{1, \ldots, n\} \) \hspace{1cm} (8.13)

\( x_{kl}^{ij} \in \{0, 1\}, \quad \forall i, k, j, l \) \hspace{1cm} (8.14)

In the next section, we present a greedy algorithm for solving the problem in polynomial time.

8.3.3 Greedy Algorithm

Algorithm 10 shows the proposed greedy algorithm. The algorithm keeps track of selected features in \( F \). The function \( h(F) \) outputs the sensor nodes that are associated with a given feature set \( F \). During each iteration, the algorithm chooses an edge with minimum cost per each new covered sensor node by that edge. The set of newly covered sensor nodes by edge \( e_{kl}^{ij} \) is computed by \( h(F \cup \{s_i, s_k\}) - h(F) \). The algorithm iterates until all sensor nodes are covered (i.e., \( h(F) = h(V) \)). For the case of \( K \)-Reliability, the algorithm will repeat the procedure on the graph without the edges that have been selected in previous rounds.

**Theorem 1.** The greedy algorithm returns a feature set of at most 1.5 times the minimum cost of any feature set covering all the sensor nodes.

**Proof.** When the greedy algorithm chooses an edge \( e_{kl}^{ij} \), imagine that it charges the
Algorithm 10 Greedy solution for MCRFS

**Input:** Reliability graph $G=(V,E,W_V,W_E)$, Degree of reliability ($K$)

**Output:** Set of minimum cost redundant features, $F$

Initialize $F ← \emptyset$.

for all $r \in 1...K$ do

Initialize $F_r ← \emptyset$, $E_r ← \emptyset$. Define $h(F_r) = \{ s_i | f_{ij} \in F_r \}$

repeat

$e_{ij}^{kl} = \arg \min_{e_{kl} \in E_r \cup \{ f_{ij} \}} (c_{ij} + w_{ij}^{kl} + c_{kl})$

$F_r ← F_r \cup \{ f_{ij}, f_{kl} \}$

$E_r ← E_r \cup \{ e_{ij}^{kl} \}$

$c_{ij} = 0; c_{kl} = 0$

until $h(F_r) = h(V)$

$F = F_r \cup F$

$E = E - E_r$

end for

return $F$

price per sensor node for that iteration to each sensor node newly covered by $e_{ij}^{kl}$.

Then the total cost of the edges chosen by the algorithm equals the total amount charged, and each element is charged only once. Since features are chosen in pairs, consider any pair of features $f_{ij}$ and $f_{kl}$ in the optimal feature set, $F^*$ associated with edge $e_{ij}^{kl}$.

During the iteration that $e_{ij}^{kl}$ is chosen, one (i.e., either $s_i$ or $s_j$) or two (both $s_i$ and $s_j$) are uncovered yet. If only one sensor node is uncovered, then the greedy algorithm would pay a cost per sensor node of at most $W_{ij}^{kl}$ charged to the sensor node being covered during this iteration ($W_{ij}^{kl} = c_{ij} + w_{ij}^{kl} + c_{kl}$). If, however, two sensor nodes are uncovered at the beginning of the iteration, then the cost per sensor node would be it
would pay a cost per sensor node of at most $W_{ij}^{kl}/2$. Summing over all sensor nodes associated with $c_{ij}^{kl}$, the total amount charged is at most $W_{ij}^{kl} + W_{ij}^{kl}/2 = 1.5 \times W_{ij}^{kl}$.

Summing over all edges in $F^{*}$ and noting that every sensor node is covered by some edge in $F^{*}$, the total amount charged to sensor nodes overall is given by

$$\sum_{W_{ij}^{kl} \in F^{*}} 1.5 \times W_{ij}^{kl} = 1.5 \times OPT$$  \hspace{1cm} (8.15)

### 8.4 Experimental Validation

In this section, we present our results on (1) the impact of anomalies on accuracy performance of activity recognition; (2) performance of individual SUs; and (3) energy efficiency of the proposed reconfiguration management framework compared to the conventional approach.

#### 8.4.1 Experimental Setup

To demonstrate the effectiveness of the proposed framework, we use the dataset with 18 nodes as described in Section 4.2.6. We constructed a network of three wearables placed on 'left wrist', 'right ankle', and 'waist'. In addition to these locations, movement data for several other network configurations are considered. These configurations resemble anomaly (e.g. misplacement) that can happen in real scenarios. We here includes sensor misplacement and displacement for our case study.

We extracted 10 statistical features as well as 10 morphological features from each
Table 8.1: Energy consumption of computing individual features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Energy (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amp</td>
<td>Amplitude of signal segment</td>
<td>16386</td>
</tr>
<tr>
<td>Med</td>
<td>Median of signal segment</td>
<td>405159</td>
</tr>
<tr>
<td>Mean</td>
<td>Mean value of signal segment</td>
<td>8126</td>
</tr>
<tr>
<td>Max</td>
<td>Maximum amp of signal segment</td>
<td>8103</td>
</tr>
<tr>
<td>Min</td>
<td>Maximum amp of signal segment</td>
<td>8129</td>
</tr>
<tr>
<td>P2P</td>
<td>Peak to peak amplitude</td>
<td>16291</td>
</tr>
<tr>
<td>Var</td>
<td>Variance of signal segment</td>
<td>38846</td>
</tr>
<tr>
<td>Std</td>
<td>Standard deviation</td>
<td>40431</td>
</tr>
<tr>
<td>RMS</td>
<td>Root mean square power</td>
<td>29705</td>
</tr>
<tr>
<td>S2E</td>
<td>Start to end value</td>
<td>83</td>
</tr>
<tr>
<td>Morph</td>
<td>Morphological Samples</td>
<td>41</td>
</tr>
</tbody>
</table>

activity trial. The features are shown in Table 8.1 where the energy consumption of computing each feature is also reported. We calculated instruction level energy consumption of MSP430 micro-controller, which is available on the TelosB motes used in our experiments. The details of energy calculations is based on an instruction level energy model provided in [96, 94]. Also, information on power consumption and specifications of the sensor nodes can be found in [97]. Morphological features were equally-spaced samples of the activity signal. Overall, we extracted 300 features from three sensors.

For allocating the communication costs in the prediction graph, we assume that the communication cost is zero for edges that connect feature of the same sensor node which runs the RMU. The communication costs of inter-node data transmission was
Table 8.2: On-body locations used for data collection in normal, and in presence of anomaly, and corresponding accuracy drops.

<table>
<thead>
<tr>
<th>ID</th>
<th>Uncertainty</th>
<th>Type</th>
<th>Drop in Recall</th>
<th>Drop in Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Right Ankle → Left Forearm</td>
<td>Misplacement</td>
<td>36.1%</td>
<td>33.2%</td>
</tr>
<tr>
<td>2</td>
<td>Left Wrist → Right Knee</td>
<td>Misplacement</td>
<td>33.4%</td>
<td>28.2%</td>
</tr>
<tr>
<td>3</td>
<td>Waist → Neck</td>
<td>Misplacement</td>
<td>25.1%</td>
<td>28.1%</td>
</tr>
<tr>
<td>4</td>
<td>Right Ankle → Right Shin</td>
<td>Displacement</td>
<td>32.1%</td>
<td>34.5%</td>
</tr>
<tr>
<td>5</td>
<td>Left Wrist → Right Forearm</td>
<td>Displacement</td>
<td>14.2%</td>
<td>16.4%</td>
</tr>
<tr>
<td>6</td>
<td>Waist → Left Hip</td>
<td>Displacement</td>
<td>15.8%</td>
<td>16.3%</td>
</tr>
</tbody>
</table>

Table 8.3: Power saving of feature selection compare to the baseline for various reliability levels ($K \in \{1, 2, 3, 4\}$).

<table>
<thead>
<tr>
<th>Power Saving/Reliability Level</th>
<th>$K=1$</th>
<th>$K=2$</th>
<th>$K=3$</th>
<th>$K=4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation Power (Misplacement)</td>
<td>97.1%</td>
<td>95.7%</td>
<td>90.6%</td>
<td>55.1%</td>
</tr>
<tr>
<td>Computation Power (Displacement)</td>
<td>97.1%</td>
<td>96.9%</td>
<td>90.2%</td>
<td>84.5%</td>
</tr>
<tr>
<td>Communication Power</td>
<td>99.0%</td>
<td>97.5%</td>
<td>97.0%</td>
<td>96.0%</td>
</tr>
</tbody>
</table>

set to be a fixed number computed based on the energy cost of transmitting on data packet using ZigBee protocol. That is, the communication power is based on ZigBee protocol specifications [98].
Table 8.4: Power consumption and accuracy of different processing units.

<table>
<thead>
<tr>
<th>Unit/Parameter</th>
<th>Precision (P)</th>
<th>Recall (R)</th>
<th>Power Consumption (µJ)</th>
<th>fp</th>
<th>tp</th>
</tr>
</thead>
<tbody>
<tr>
<td>SU (DT)</td>
<td>99.1%</td>
<td>99.3%</td>
<td>250.045</td>
<td>0.01</td>
<td>0.99</td>
</tr>
<tr>
<td>MU1 (KNN)</td>
<td>98.1%</td>
<td>97.9%</td>
<td>4630.605</td>
<td>0.02</td>
<td>0.98</td>
</tr>
<tr>
<td>MU2 (DT)</td>
<td>86.6%</td>
<td>86.8%</td>
<td>1304.097</td>
<td>0.13</td>
<td>0.87</td>
</tr>
</tbody>
</table>

8.4.2 Impact of Anomaly

To show the impact of anomaly on the performance of networked wearables, we used the collected dataset for activity recognition where we develop a machine learning classifier to detect physical activities. Given that $k$-Nearest Neighbor ($k$NN) classification has shown promising results in activity recognition [55], we chose $k$NN for activity classification. First, we developed a machine learning classifier using the perfect data, and then tested this model using ten-fold cross validation, which provided us with an average accuracy of 97.8% for detecting the 30 activities. Later, we test the activity recognition model on data with different types of anomalies as listed in Table 8.2. In each case, the assumption is that there is just one anomaly in the network. Table 8.2 shows the amount of accuracy drop due to these anomaly occurrences, namely sensor misplacement and sensor displacement. As shown in Table 8.2, the accuracy of activity recognition drops by 22.6% on average.
8.4.3 Feature Selection Result

To assess the performance of the anomaly screening unit, we constructed two prediction graphs with 300 vertices associated with the extracted features for two anomaly types. The inter-feature correlations are needed for both perfect data and data in presence of anomaly (i.e. one sensor is misplaced) to build each prediction graph.

The number of edges in the prediction graphs is directly dependent on the value of $\Delta$. Clearly, the number of edges decreases as the threshold increases. We experimentally determined a range of $3.5 \leq \Delta \leq 3.9$ to be an acceptable range. If we choose $\Delta$ outside of these range the graph would be either sparse, or contain many irrelevant features. As the level of reliability increased from $K=1$ to $K=4$, the number of selected features increased from 3 to 9.

To compute the accuracy performance of the screening units that reside in our anomaly screening module, we only considered single-anomaly as listed in Table 8.2. We used three classification algorithms including Decision Tree (DT), ($k$NN), and support vector machine (SVM) for this analysis. The accuracy of screening units ranged from 73.7% to 100% using different classifiers and different levels of reliability as shown in Fig. 8.4. The average accuracy for misplacement, and displacement cases are 91.1%, and 98.5% using a decision tree classifier.
Figure 8.4: Accuracy of SUs using DT, kNN, and SVM classifiers respectively; The results are shown for both misplacement (a) and displacement (b).

8.4.4 Energy Consumption Comparison

Table 8.3 shows the amount of energy savings obtained using our power-aware feature selection technique compare to the baseline case with no optimization. The energy savings ranged from 55.1% for the highest reliability level ($K=4$) to 97.1% for the lowest reliability level ($K=1$) in terms of feature computation power. The communication power saving is also based on the percentage of features that are needed to transmit using the proposed framework compared to the baseline scenario.

In this section, energy analysis of the propose framework compared to the conventional approach is presented. Given that we focused on two anomaly types, our framework consists of two SUs and two MUs one for each anomaly type. Table 8.4
shows the energy consumption, and accuracy of anomaly detection and mitigation units that are used in our system. The MU units in Table 8.4 are on-body node localization models which are used to mitigate the effects of misplacement, and displacement.

We measured the total computation energy of both approaches as a function of \( \alpha \), the activation rate of the MUs. In this comparison, we only considered energy consumption of signal processing and feature extraction as other energy components are used in both the conventional model and the proposed framework.

![](image)

**Figure 8.5:** Energy consumption comparison of the proposed framework and the conventional approach using kNN (a) and DT (b) classifiers.

The energy consumption of the two approaches (i.e., the conventional and the proposed) is shown in Fig. 8.5a and Fig. 8.5b. With \( \alpha = 0.1 \), the amount of power
savings of our framework is 88% and 69% for kNN and DT classifiers respectively. It is reasonable to assume $\alpha = 0.1$ as an uncertainty rate [92]. In short, our system outperforms the conventional model for $\alpha \leq 0.93$ using kNN, and $\alpha \leq 0.81$ using DT, as shown in Fig. 8.5b. We note that for extremely larger $\alpha$ (e.g. $\alpha > 0.93$ for kNN and $\alpha > 0.81$ for DT), the conventional approach outperforms ours. This situation, however, is unrealistic because anomalies are sparse in time that they occur at most 10% of the times.

8.5 Summary

We proposed an energy-efficient framework which is used to manage occurrences of anomalies in mobile sensors and take corresponding actions. A feature selection algorithm was developed to exploit correlation variation of pair of features. The energy saving of framework is 88.2% and 69.7% compared to conventional methods considering anomaly occurrence of 10% for kNN-based, and DT-based MUs respectively. There is a trade off between power consumption and accuracy of MUs, the more accurate classifier, the lower amount of power saving. simple SUs detect anomaly types with 98.3% accuracy on average. We focused on activity recognition as our pilot application and performed our analysis on motion sensors such as accelerometers and gyroscope sensors.
CHAPTER 9. CONCLUSION AND FUTURE DIRECTIONS

In this dissertation, we proposed advanced algorithms to cope with the current limitations of M-health systems in their dynamically changing environments. We revisited different components of the general machine learning process chain and equip the system with novel submodules for reconfiguration. To reach this goal, we used several inter-related organizing principles in the design process. First, we proposed a novel design framework to support autonomous reconfiguration of M-health systems under different kinds of data variations. In the framework, context and configuration change in the system is monitored in the real-time execution state, and when sufficient change is detected, a procedure is activated in the reconfiguration state to autonomously rebuild the computational model. As part of the framework, we developed four data mapping algorithms for extracting knowledge from training data in the reconfiguration state. These algorithms are designed combining effective methods for signal similarity identification, motif discovery, clustering, and manifold learning.

Secondly, we explored the use of active learning for reconfiguration of mobile health monitoring systems. We mainly focused on designing an expert selection algorithm which can be used in different real-world scenarios with heterogeneous set of experts. The expert selection module is incorporated as a core part of the expert
management unit within the active learning architecture called Co-MEAL. The expert management unit cooperates with the query strategy module and the transfer learning (i.e. learner initialization) module to further decrease the cost of the active learning process. To provide more efficient annotation, the Co-MEAL architecture is designed to allow collaboration among experts in order to reduce annotation cost and enhance knowledge of inexpensive experts for future data labeling.

Thirdly, we proposed a novel deep architecture for HAR, capable of building generalizable algorithms robust against data variations. This architecture has the potential to uncover features that are tied to the dynamics of human motion production, from simple motion encoding in lower layers to more complex motion dynamics in upper layers. The complexity of a deep neural network can help mitigate the effects of variations in the data such as walking with different paces. Furthermore, the architecture also capture the relation between consecutive data segments. The architecture includes an active learning module; whenever the system faces uncertainty (i.e. the ID score for incoming samples is high) in the real-time execution, the active learning phase is triggered to update the deep architecture.

Finally, to manage how the system should response to data variations, we propose a novel energy-efficient reconfiguration management unit to cooperate with all reconfiguration units. Our architecture uses computationally simple screening units for detection of sparse uncertainties (e.g. configuration change). This architecture is
inspired by power management techniques in design of digital systems. We need to be able to design low complexity and energy-efficient algorithms for this unit, because that it should be running consistently on mobile devices.

The autonomous reconfiguration framework can be extended in several directions to increase its scalability. First, it can be extended by adding a context/configuration identification submodule which can find the most relevant source domain(s) to the current operating setting before the search process. Second, it would provide more accurate to develop algorithms quantifying the capability of different axis of data for each class of data. This can help us with more exact data mapping, and also decrease the similarity calculation burden. Furthermore, parallel implementation of data mapping for multi-user systems could be another direction. All of the mapping algorithms have an intrinsic parallel nature and it is possible to build parallel algorithms.

In Chapter 6, we assumed that there is no time-dependency between data instances. Future work could develop a query strategy method in such a way that it reduces the cost of data annotation by using the information on time-dependency of different data instances. Specifically, development of methods to group data instances based on their closeness (time-wise) should reduce the cost of data annotation. The work in Chapter 7 can be extended in two directions. Designing deep architectures robust against more complex heterogeneity situations such as mis-orientation and misplacement of sensors is one direction. In another direction, future work could
explore the suitability of integrated methods for transfer and active learning, instead of having them sequentially in the system development process.
BIBLIOGRAPHY


[5] Ramin Fallahzadeh, Mahdi Pedram, Ramyar Saeedi, Bahman Sadeghi, Michael


ized medicine: risk prediction, targeted therapies and mobile health technology. 


[21] Niloofar Hezarjaribi, Ramin Fallahzadeh, and Hassan Ghasemzadeh. A machine learning approach for medication adherence monitoring using body-worn


[26] Vincent Wenchen Zheng, Derek Hao Hu, and Qiang Yang. Cross-domain activ-


[38] Thomas Plötz, Nils Y Hammerla, and Patrick Olivier. Feature learning for ac-


[54] Hamed R Bonab and Fazli Can. A theoretical framework on the ideal number


[68] Billur Barshan and Murat Cihan Yüksel. Recognizing daily and sports activities
in two open source machine learning environments using body-worn sensor units.


[73] Abdullah Mueen, Eamonn Keogh, Qiang Zhu, Sydney Cash, and Brandon West-


[79] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron


[85] Ramyar Saeedi, Hassan Ghasemzadeh, and Assefaw Gebremedhin. Transfer


[90] Hesam Sagha, Alberto Calatroni, Jose del R Millan, Daniel Roggen, Gerhard Troster, and Ricardo Chavarriaga. Robust activity recognition combining


