LEARNING TO ADAPT TO SENSOR CHANGES AND FAILURES

by

Yuan Shi

A Dissertation Presented to the

FACULTY OF THE USC GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(COMPUTER SCIENCE)

August 2019

Dedication

To my father Xiangyun Shi and my mother Hongying Zhang for their love and support.

Acknowledgments

I would like to thank Prof. Craig A. Knoblock for advising me in my PhD program at the University of Southern California. This thesis is a product of my research work under his supervision. During the course of my PhD program, he supported me financially, academically and emotionally. He also gave me great advice on identifying research problems, developing novel approaches, writing papers and giving presentations, among many other skill developments. Prof. Knoblock is also a role model for me in many aspects of life.

I would also like to thank Prof. T. K. Satish Kumar, whom I consider as my secondary advisor. Prof. Kumar helped me with research, publications and presentations. His guidance kept me on the right track so that I could successfully finish writing my thesis in time and meet high quality standards.

I would like to thank Prof. Fei Sha, who served as an advisor during the early phase of my PhD program. Prof. Sha set a high standard on my research work and advised me extensively on critical thinking, problem solving and academic writing.

I would also like to extend many thanks to my other thesis committee members, Prof. Yan Liu and Prof. Daniel Edmund O'Leary, for perusing the entire thesis and giving me critical comments and guidance. Thanks to Prof. Cyrus Shahabi and Prof. Bhaskar Krishnamachari for serving as committee members of my qualifying exam and helping me refine the research topics and methodologies.

Finally, I want to express my thanks to all my collaborators: Boqing Gong, Wenzhe Li, Aurelien Bellet, Prof. Kristen Grauman, Ang Li, Kavya Sethuraman, Fiona Khatana, Prof. Avi Pfeffer, Curt Wu, Gerald Fry, Kenneth Lu, Stephen Marotta and Mike Reposa, among others. I learned a lot from their distinctive perspectives and insights into solving research problems. Thanks to them for creative discussions, supportive comments and interesting thoughts.

This material is based upon work supported by the United States Air Force and the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-16-C-0045. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force and DARPA.

The U.S. Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

Contents

List of Tables

List of Figures

Abstract

Many software systems run on long-lifespan platforms that operate in diverse and dynamic environments. As a result, significant time and effort are spent manually adapting software to operate effectively when hardware, resources and external devices change. If software systems could automatically adapt to these changes, it would significantly reduce the maintenance cost and enable more rapid upgrade. As an important step towards building such long-lived, survivable software systems, we study the problem of how to automatically adapt to changes and failures in sensors.

We address several adaptation scenarios, including adaptation to individual sensor failure, compound sensor failure, individual sensor change, and compound sensor change. We develop two levels of adaptation approaches: sensor-level adaptation that reconstructs original sensor values, and model-level adaptation that directly adapts machine learning models built on sensor data. Sensor-level adaptation is based on preserving sensor relationships after adaptation, while model-level adaptation maps sensor data into a discriminative feature space that is invariant with respect to changes.

Compared to existing work, our adaptation approaches have the following novel capabilities: 1) adaptation to new sensors even when there is no overlapping period between new and old sensors; 2) efficient adaptation by leveraging sensor-specific transformations derived from sensor data; 3) scaling to a large number of sensors; 4) learning robust adaptation functions by leveraging spatial and temporal information of sensors; and 5) estimating the quality of adaptation.

Additionally, we present a constraint-based learning framework that performs joint sensor failure detection and adaptation by leveraging sensor relationships. Our framework learns sensor relationships from historical data and expresses them as a set of constraints. These constraints then provide a joint view for detection and adaptation: detection checks which constraints are violated, and adaptation reconstructs failed sensor values. Our framework is capable of handling multi-sensor failures which are challenging for existing methods.

To validate our approaches, we conduct empirical studies on sensor data from the weather and UUV (Unmanned Underwater Vehicle) domains. The results show that our approaches can automatically detect and adapt to sensor changes and failures with higher accuracy and robustness compared to other alternative approaches.

Chapter 1

Introduction

1.1 Motivation

An increasing number of applications require long-term autonomy of software systems and their capability to operate in dynamic environments. Maintaining the quality, durability, and performance of these software systems is very challenging and labor-intensive. Failure to effectively and timely adapt to hardware and resource changes can result in technically inferior and potentially vulnerable systems [62]. For example, software systems based on sensor data can suffer from sensor failures or changes caused by environmental conditions and technical errors [37]. Occasionally, such failures can cause severe safety issues, e.g., faulty sensor data caused the crash of a Lion Air Flight 610, killing all 189 people on board.[1] If software systems could automatically detect sensor failures, these types of catastrophes could be avoided. In addition, if software systems could adapt to sensor failures and changes, we could significantly reduce the time and effort required for software maintenance and promote the long-term use of quality software on platforms that are under continuous change.

As an important step towards building such long-lived, survivable software systems, we study the problem of how to automatically adapt to failures and changes in sensors. Sensor failure happens when a sensor stops generating normal sensor

[1]`https://www.cnn.com/2018/11/28/asia/lion-air-preliminary-report-intl/index.html`

values; sensor change happens when a sensor gets replaced by other sensor(s). Our goal is to build machine-learning-based *adapters* that can largely reduce the effect of sensor failures or changes on upper-layer software. We believe the solutions to this problem can have broad impact since there is an increasing volume of sensors that are deployed in real-world systems [36, 52].

1.2 Sensor-level and Model-level Adaptation

When sensor failures or changes happen, ideally, we would like an adapter to recover the original sensor values. We consider this as *sensor-level adaptation*, which aims at learning a reconstruction function by leveraging the relationships among working sensors. The underlying assumption is that sensor values from a subset of sensors are correlated, which is often the case in real-world systems [39, 52]. For example, temperature, humidity and dew point measured by weather sensors are inter-correlated [70], and any two of them can be used to predict the third one well. If sensor-level adaptation can accurately reconstruct the original sensor values, then the reconstructed values can be directly input to upper-layer software, without additional change to the software. Fig. 1.1 shows an example of weather sensors, continuously generating timestamp, latitude, longitude, pressure and temperature values. In the second and third rows, the temperature sensor fails to work properly. To address this, sensor-level adaptation reconstructs the original temperature values by leveraging sensor values from the remaining sensors and possibly new sensors.

However, sensor-level adaptation can be challenging or even impossible when the remaining sensors are weakly correlated with the sensors we would like to

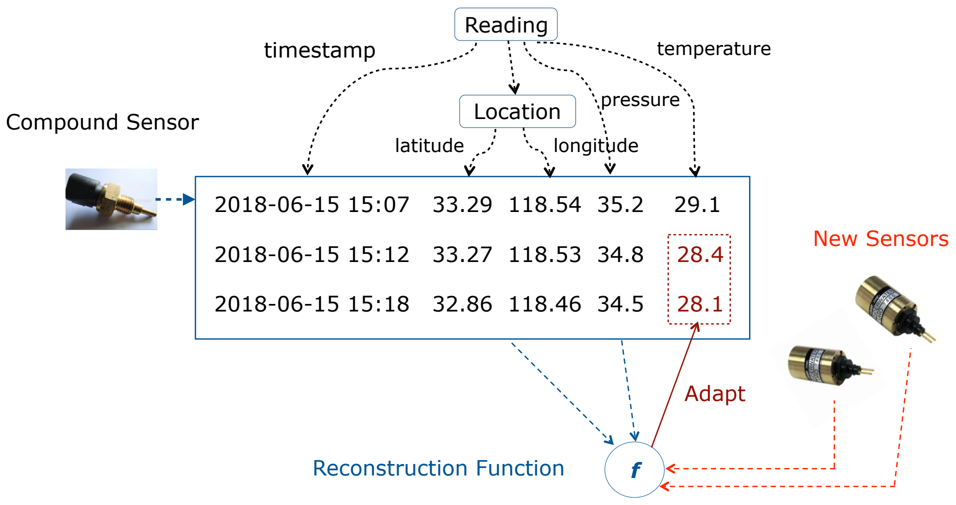


Figure 1.1: Example of a compound weather sensor that consists of three individual sensors. The reconstruction function f reconstructs failed temperature values from the two remaining working sensors (blue arrows) and two new sensors (red arrows).

reconstruct. In such a case, we consider another level of adaptation called *model-level adaptation*. Instead of reconstructing the original sensor values, model-level adaptation aims at directly adapting software components that are built on the original sensor values. In particular, we study how to automatically adapt machine learning models (e.g., classifiers) built on sensor values. Model-level adaptation can be viewed as an instance of *domain adaptation* or *transfer learning* [34, 77] which adapts a machine learning model from a *source domain* to a similar but different *target domain*. We can view sensor readings before and after sensor failures/changes as the source and target domains, respectively. It is easy to see that model-level adaptation may be still feasible when sensor-level adaptation is not. As an extreme example, if the model does not rely on the failed or changed sensor, then model-level adaptation is always feasible - simply re-using the model.

1.3 Adaptation Scenarios

We examine four adaptation scenarios. These scenarios rely on the notion of a *compound sensor*. A compound sensor consists of a set of individual or component sensors, each measuring a certain type of signal. For example, a weather station can be viewed as a compound sensor composed of individual sensors measuring temperature, humidity, dew point, wind speed and wind gust, etc. For each scenario, we discuss both sensor-level adaptation and model-level adaptation. The four adaptation scenarios are described as follows (see Fig. 1.2 for an illustration).

- Individual Sensor Failure: some but not all of the individual sensors in a compound sensor fail to produce normal values. The failed individual sensors may simply stop working or produce abnormal values that cannot be used.

- Compound Sensor Failure: all individual sensors in a compound sensor fail to produce normal values.

- Individual Sensor Change: some but not all of the individual sensors in a compound sensor are replaced by another set of individual sensors. This corresponds to the cases where new individual sensors are plugged in manually or automatically when sensor failures or sensor upgrades occur. Throughout the thesis, we refer to sensors that are replaced by new sensors as *replaced sensors*. Typically, sensor values of replaced sensors no longer exist after sensor replacement. Therefore, no overlapping period exists between the replaced and new sensors. This makes the adaptation very challenging because new sensors can produce significantly different values[2] compared to replaced sensors. Furthermore, new sensors may measure additional types of signals that do not exist before the sensor change.

- Compound Sensor Change: the entire compound sensor is replaced by a new compound sensor. This happens in practice for the reason that replacing the compound sensor is technically easier than replacing individual sensors in certain systems. This scenario is more challenging than Individual Sensor Change, since no individual sensor from the compound sensor can be used to calibrate new sensors.

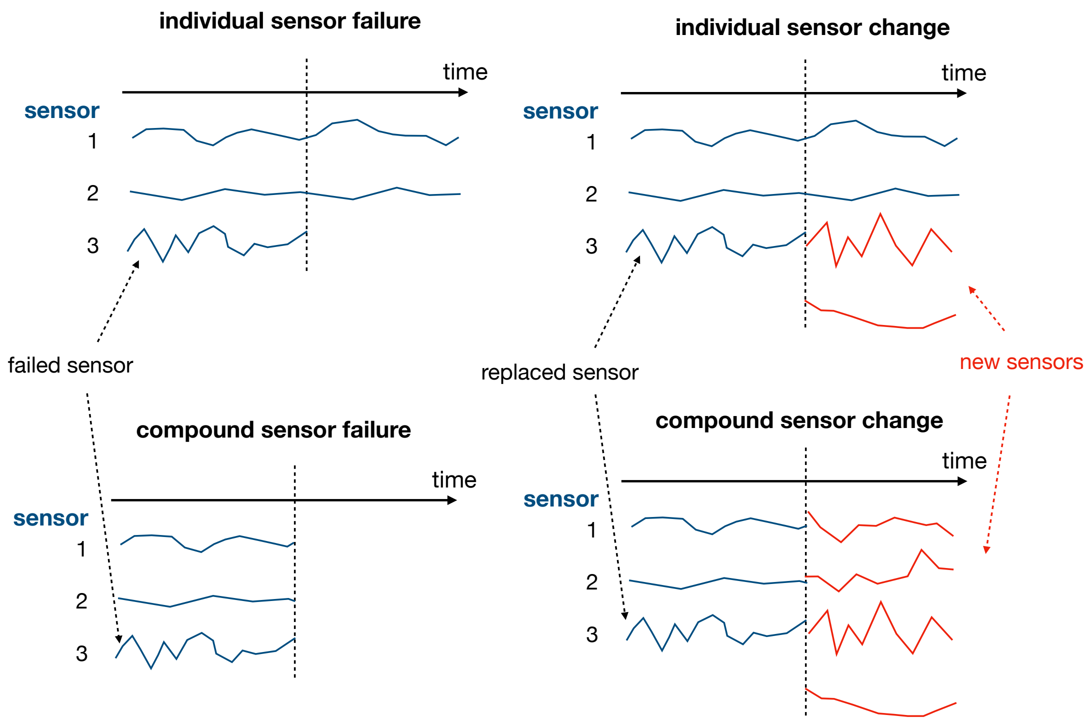


Figure 1.2: Four scenarios of sensor failures and changes.

[2]New sensors may also produce sensor values in different formats, which is not the focus of this thesis.

We propose a series of adaptation approaches to address these scenarios. For sensor-level adaptation, our approaches learn reconstruction functions that reconstruct original sensor values from all working sensors including the new ones. The general methodology behind our approaches is to preserve sensor relationships after reconstruction, where sensor relationships can be derived from historical sensor values. Different from existing work on handling sensor failures and changes [39, 36, 37], our approaches have two unique features. First and most importantly, our approaches are capable of exploiting new sensors although there is no overlapping period between the new sensors and the replaced ones. To the best of our knowledge, this is the first work on such problem settings. Previous work that take into account new sensors or input features [60, 103] invariably require ground-truth labels. Second, our reconstruction functions can leverage sensor-specific transformations learned from historical sensor data. This reduces the number of parameters in the reconstruction functions, which not only leads to better interpretability of the learned functions but also makes the learning process more efficient. The latter property enables more rapid adaptation during sensor changes. Part of this work was published in our earlier paper [87].

For model-level adaptation, we propose a general domain adaptation approach that learns a feature space where data in the source domain (before sensor change) and target domain (after sensor change) are similarly distributed. Additionally, our approach enforces the feature space discriminative, by optimizing an information-theoretic metric as a proxy to the classification error on the target domain. Compared to existing domain adaptation work [77], our approach can effectively adapt to new sensors in an *unsupervised* way. Part of this work was also published in our earlier paper [88].

We further improve our adaptation approaches in three directions. First, we exploit the fact that additional information about sensors may be available. In particular, we consider cases where spatial and temporal information about sensors can be easily obtained [25, 36]. For example, when the temperature sensor fails, a weather station may immediately access the same sensor from a nearby station whose location is known. Also, the exact timestamps of the two temperature sensors are often available, which can be used to calibrate their signals. We propose an approach to learn calibration functions that can align signals from different sensors based on their spatial and temporal information. Once such calibration functions are learned, they can be used to pre-calibrate signals from new sensors before learning actual adaptation functions. Such calibration makes new sensors better aligned to replaced sensors and can improve the robustness and accuracy of the learned adaptation functions.

Second, we scale our approaches to a large number of sensors when dealing with sensor changes. Our method selects a subset of important sensors based on correlations among sensor values, which can significantly reduce the overfitting to noisy values as well as the overall computational cost. This also enables our adaptation approaches to continuously exploit new sensors in an open environment.

Third, we propose a method to dynamically estimate the adaptation quality, which enables upper-layer software components to determine whether or not to accept an adaptation. This also provides a way to select an optimal adaptation strategy when multiple adaptation strategies exist.

In the above adaptation approaches, we assume that sensor failures or changes are known. In practice, however, such failures or changes need to be detected first. Following our adaptation methodology, we can also use sensor relationships for detection. To this end, we propose a novel computational framework that performs detection and adaptation jointly. Our framework extracts sensor relationships from historical data in the form of constraints, and uses these constraints for both detection and adaptation.

- **Detection** checks each constraint, and identifies a sensor failure if one or more constraints are violated. It then infers the likely failed sensor(s) from the violated constraints.

- **Adaptation** reconstructs the failed sensor values from the remaining working sensor values by using the set of constraints. Tighter constraints correspond to more accurate reconstruction relationships.

Compared to existing work [10, 53, 17, 79], our framework is capable of detecting and adapting to *multi-sensor failures* where multiple sensors fail simultaneously.

To validate our proposed approaches, we have conducted empirical study on weather sensor data[3] and UUV (Unmanned Underwater Vehicle) data. Experimental results show that our approaches can automatically adapt to sensor failures and changes, with higher accuracies than baseline methods.

To summarize, we have proposed a series of approaches for automatically adapting to sensor failures and changes. Our approaches have the following novel capabilities.

- exploiting two levels of adaptation: sensor-level and model-level

- adaptation to new sensors when there is no overlapping period between the new sensors and the replaced sensors

- efficient adaptation by leveraging sensor-specific transformations derived from historical sensor data

[3]`https://www.wunderground.com/`

- learning how to adapt robustly and accurately by leveraging spatial and temporal information about sensors
- scaling to a large number of new sensors
- estimating the quality of adaptation
- performing joint detection and adaptation for sensor failures via a constraint-based framework

The software and datasets associated with this thesis can be accessed at our Github repository[4].

Thesis Statement. This thesis proposes a series of machine learning approaches for automatically adapting to sensor failures and changes. These approaches exploit sensor relationships and can address failures/changes in both individual sensors and compound sensors.

This thesis is of highest relevance to researchers and practitioners working in the areas of Software Systems, Internet of Things and Machine Learning.

[4]https://github.com/usc-isi-i2/sensor-adaptation. For any questions or comments, please contact the author at yuanshi@usc.edu.

Chapter 2

Settings and Notations

2.1 Problem Settings

We study the general setting of sensor failures and changes in the context of a compound sensor. Imagine that we have a compound sensor consisting of multiple individual or component sensors. For example, a compound sensor can be a weather station containing several weather sensors measuring temperature, dew point, wind speed, etc. An instant of time at which some sensor(s) fail or are replaced by new sensors is called a change point.[1] As described in Section 1.3, we consider four scenarios: individual sensor failure, compound sensor failure, individual sensor change, and compound sensor change.

The sensor change scenarios are more challenging than the sensor failure scenarios since there is no overlapping period between the new sensors and the replaced sensors. In individual sensor change, the remaining sensors are the key to link the information between the new sensors and the replaced sensors. We call the remaining sensors as *reference sensors*. Intuitively, if the reference sensors are correlated with both the new sensors and the replaced sensors, they can be helpful for reconstructing the replaced sensor values from the new sensor values. For compound sensor change, however, there are no reference sensors from the compound sensor because all sensors are replaced. In this scenario, adaptation to new sensors

[1]We only address a single change point. Repeated invocation of our methods naturally handles multiple change points as well.

is very challenging or even impossible. To enable reasonable adaptation, therefore, we assume that we have access to some reference sensors outside the compound sensor. For example, in the context of weather stations, we can use sensors in other stations as reference sensors.

Using the notion of reference sensors, the four scenarios can be viewed in a *unified* way:

- reference sensors always work properly
- replaced sensors are replaced by new sensors at the change point. In sensor failure scenarios, there are no new sensors.

Fig. 2.1 visualizes this unified view and the corresponding notations (explained below).

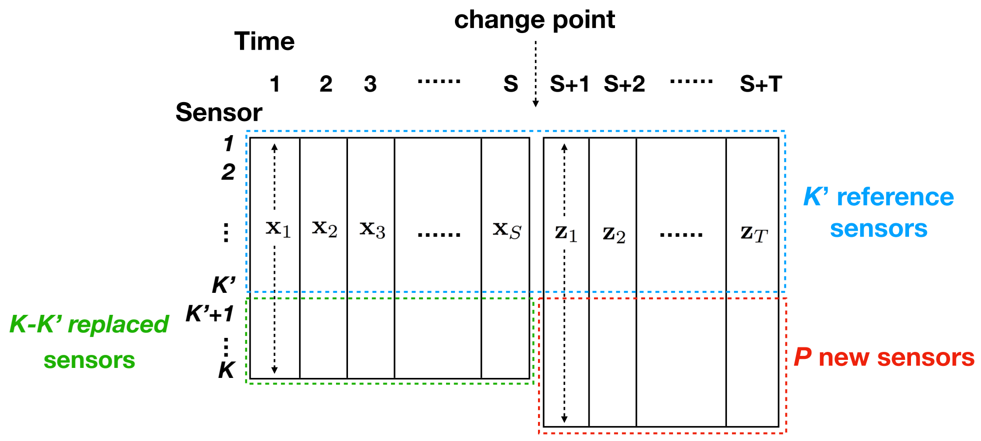

change point
Time
1 2 3 ...... S S+1 S+2 ...... S+T
Sensor
1
2
⋮
K'
K'+1
⋮
K
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ ...... $\mathbf{x}_S$
$\mathbf{z}_1$ $\mathbf{z}_2$ ...... $\mathbf{z}_T$
K' reference sensors
K-K' replaced sensors
P new sensors

Figure 2.1: Settings and notations for sensor failures and changes.

2.2 Notations

Suppose we are given K individual sensors, among which K' sensors are reference sensors. We assume that

- All sensors generate sensor values at fixed time intervals, and sensor values are temporally aligned;
- Sensor values start at time 1. At time $S+1$, $K-K'$ sensors are replaced by $P \geq 0$ new sensors ($P=0$ corresponds to sensor failure). We have sensor values until time $S+T$;
- There is only a single change point, i.e., time $S+1$, and it is already given. (Detecting the change point will be addressed in Chapter 6.)

Let $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_S$ be sensor values before the change point, where $\mathbf{x}_s \in \mathcal{R}^K$ represents sensor values at time $s \in \{1, 2, \cdots, S\}$, and $\mathbf{x}_{s,k}$ represents the corresponding sensor value from sensor $k \in \{1, 2, \cdots, K\}$. Additionally, let the replaced sensors be sensors $K'+1, K'+2, \cdots, K$. Let $\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_T$ denote sensor values after the change point, where $\mathbf{z}_t \in \mathcal{R}^{K'+P}$ represents sensor values at time $S+t$, for $t \in \{1, 2, \cdots, T\}$. Note that we use s to index $\mathbf{x}$ and t to index $\mathbf{z}$. Based on this setting, $\{\mathbf{x}_{s,k}\}$ and $\{\mathbf{z}_{t,k}\}$ for $k \in \{1, 2, \cdots, K'\}$ represent sensor values of reference sensors, and $\{\mathbf{z}_{t,k}\}$, for $k \in \{K'+1, K'+2, \cdots, K'+P\}$ represent sensor values of the P new sensors. Fig. 2.1 illustrates the above notations.

In the following chapters, we often refer to sensor values before the change point as the *source* domain, and sensor values after the change point as the *target* domain. Similar notions are used in the domain adaptation and transfer learning communities [77].

2.3 Datasets

In the following chapters, we conduct empirical studies on sensor data from two domains.

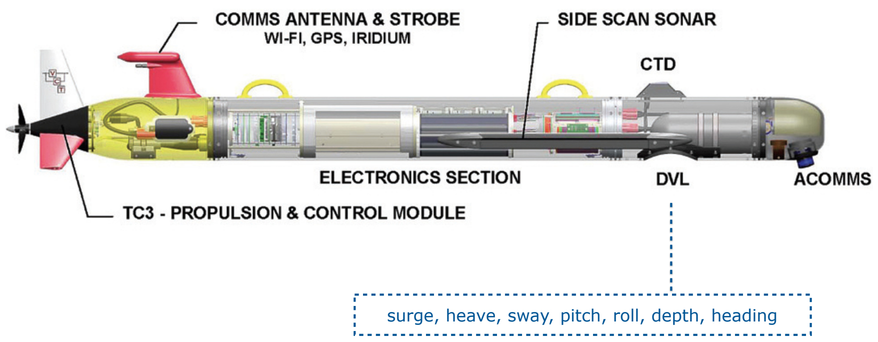


Figure 2.2: UUV Sensors (RPM, Waterspeed, DVL).

- **Weather data**: the dataset consists of weather sensor data collected from Weather Underground[2]. The dataset involves a number of personal weather stations, and each station (compound sensor) contains a set of individual sensors including temperature, dew point, humidity, wind speed, wind gust, pressure, etc. These weather stations are selected from a set of clusters/regions (e.g., Los Angeles, San Francisco, Austin, Chicago, etc.), each with 3 stations. Stations within a cluster tend to produce more similar sensor values than those across clusters. Sensor values are sampled every 5 or 10 minutes. We temporally align sensor values as a preprocessing step.

- **UUV data**: the dataset is collected by letting a UUV travel from a starting point to an end point in a simulated environment. The UUV contains propeller RPM sensor, waterspeed sensor and a compound sensor called Doppler Velocity Log (DVL) sensor. The DVL sensor consists of seven individual sensors including surge, heave, sway, pitch, roll, depth, and heading. Figure 2.2

[2]`https://www.wunderground.com/`

shows the locations of these sensors on a UUV. Each sensor produces a sensor value every second. We simulate 20 trips and collect sensor values at each second. The trajectory of the UUV varies in each trip due to different starting/end points and water currents. The total number of samples in each trip varies between 500 and 2000.

Chapter 3

Adaptation to Sensor Failures

In this chapter, we address individual sensor failures and present sensor-level and model-level adaptation approaches. The problem setting follows Chapter 2.

3.1 Sensor-level Adaptation

For sensor-level adaptation, we would like to not only learn accurate reconstruction functions to recover failed sensor values but also discover meaningful transformations that can be applied to sensor values. We propose to gather such transformations into a library and later apply them to sensor data whose sensor types are known or can be recognized. This is particularly useful when we deal with sensor changes, since these transformations help generate meaningful feature representations for new sensors. Motivated by the above arguments, we adopt a nonlinear regression approach called Fast Function Extraction (FFX) [73], which is capable of learning nonlinear functions in compact forms. Sensor-specific transformations can be easily derived from these learned forms.

Sensor-level adaptation attempts to reconstruct failed sensor values after time S based on the reference sensors. The underlying assumption is that sensor values in real-world systems are often correlated [41].

To reconstruct the failed sensors $K'+1, K'+2, \cdots, K$, we learn a separate reconstruction function for each one. For example, we can learn the following function to reconstruct sensor K from reference sensors $1, 2, \cdots, K'$ based on sensor values before the sensor failure:

$$f(\mathbf{x}_{s,1}, \mathbf{x}_{s,2}, \cdots, \mathbf{x}_{s,K'}) \rightarrow \mathbf{x}_{s,K} \tag{3.1}$$

Once f is learned, it can be used to reconstruct sensor K's values at time $S+t$ by computing $f(\mathbf{z}_{t,1}, \mathbf{z}_{t,2}, \cdots, \mathbf{z}_{t,K'})$.

Learning Eq.(3.1) is a classical regression problem [43], where a number of regression methods such as linear regression [43], kernel ridge regression [75] and neural networks [4] can be applied. Previous work [37, 36, 39] explored linear relationships among sensors, which lacks the power to model the nonlinearity of sensor data. In our implementation, we explore two regression methods, feedforward neural networks [4] and Fast Function Extraction (FFX) [73]. We explore FFX because of its capability of learning compact nonlinear function forms efficiently and leveraging sensor-specific transformations derived from domain knowledge. Experimental results below show that FFX performs comparably to neural networks with significantly fewer parameters.

Specifically, to learn a function that maps a vector $\mathbf{x}$ to real value u, FFX uses a linear form

$$u = w_0 + \sum_{i}^{N_h} w_i h_i(\mathbf{x}) \tag{3.2}$$

where $\{h_i()\}$ are pre-defined basis functions, and $\{w_i\}$ are linear coefficients to learn. FFX first generates a massive set of basis functions based on various nonlinear transformations, and then applies a machine learning technique called pathwise regularized learning [106] to efficiently select a small set of most useful basis functions. As a result, FFX is able to learn compact functions that are more interpretable than black-box methods like neural networks [73]. At the same time, FFX can learn highly nonlinear functions and achieve comparable performance to neural networks [73]. This gives FFX an attractive benefit: it helps identify useful and interpretable basis functions associated with sensor types. These sensor-specific transformations can be stored in a library and later applied to corresponding sensors to extract meaningful features.

Note that when generating basis functions in FFX, it is easy to incorporate sensor-specific transformations derived from domain knowledge. For example, in the weather domain, the relationship between temperature, dew point and humidity has been well studied [2]. Let TP, DP and HU denote temperature (°C), dew point (°C) and humidity (%), respectively. HU can be approximated by TP and DP in the following way [2]

$$HU = 100 \frac{\exp(\frac{aDP}{b+DP})}{\exp(\frac{aTP}{b+TP})} \tag{3.3}$$

where a and b are constants. Based on this domain knowledge, we can treat sensor-specific transformations $\exp(\frac{aDP}{b+DP})$, $\exp(\frac{aTP}{b+TP})$ and $\frac{\exp(\frac{aDP}{b+DP})}{\exp(\frac{aTP}{b+TP})}$ as basis functions in FFX, which potentially leads to more compact functions.

In our implementation, we use the following basis functions: $ax + b, x^a, \exp(x), \log(x), \min(0, x-a)$, where a and b are coefficients with many possible values. Note that these basis functions can be applied recursively to a single variable (e.g, $\exp(2x+3)$) or interacting variables (e.g., $x^2 \log(y)$). Moreover, by using the rational function technique [73], we enable our method to incorporate basis functions such as $\frac{\exp(\frac{ax}{b+x})}{\exp(\frac{ay}{b+y})}$.

Compound Sensor Failures. For compound sensor failures, there are no reference sensors from the compound sensor itself because all sensors fail. In this scenario, adaptation is very challenging or even impossible. To enable reasonable adaptation, we assume that we have access to some reference sensors outside the compound sensor. For example, in the context of weather stations, we can use sensors in other stations as reference sensors. This reduces to the scenario of individual sensor failure although reference sensors tend to correlate less with the failed sensors. We can then apply the same adaptation approach developed for individual sensor failures.

3.1.1 Experiments

We evaluate our adaptation approach on sensor data from the weather and UUV domains. We compare the following methods:

- **Replace**: non-adaptation method that substitutes each failed sensor with a sensor that has the closest mean and variance in sensor values. The closeness is measured on sensor values before the sensor failure;
- **Refer-Neu**: our adaptation approach that reconstructs sensor values of failed sensors using reference sensors. The reconstruction function is learned via neural networks [4];
- **Refer-FFX**: our adaptation approach with the reconstruction function learned via FFX [73];

Reconstruction errors of each method are measured by RMSE (Root Mean Square Error) between reconstructed sensor values and the ground truth.

Results on weather data

Following the dataset description in Chapter 2.3, we use 30 weather stations from 10 clusters and generate random pairs across clusters. We generate each random pair in the following way: 1) randomly select two clusters/regions; 2) randomly select one station from the first cluster (denoted as station A) and one station from the second cluster (denoted as station B). We generate 100 random pairs across 8 clusters. For each pair, we use one year (2016) of sensor data as the source domain and one year (2017) of sensor data as the target domain.

Individual sensor failure. For each random pair, we treat each sensor in station A as the failed sensor and the remaining sensors in station A plus all sensors in station B as reference sensors. Table 3.1 reports reconstruction errors on each failed sensor. We can see that in all cases, **Refer-Neu** and **Refer-FFX** perform significantly better than **Replace**, demonstrating that sensor relationships are very helpful in reconstructing failed sensor values. The relatively poor performance of **Replace** reveals that the same sensor from nearby stations (i.e., within a cluster) can generate sensor values with significant deviation. The reconstruction errors on wind speed, wind gust and pressure are relatively large because these sensor values are not strongly correlated with other sensor values. We also observe that **Refer-Neu** and **Refer-FFX** perform comparably, although **Refer-Neu** uses more parameters.

Compound sensor failure. We treat all sensors in station A as failed sensors, and all sensors in station B as reference sensors. Table 3.2 reports reconstruction

Table 3.1: Reconstruction errors (RMSE) on weather data for individual sensor failures. Each entry shows the average reconstruction error and the corresponding standard error. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Failed Sensor	**Replace**	**Refer-Neu**	**Refer-FFX**
temperature (°F)	3.94 ± 0.024	**0.58 ± 0.013**	**0.61 ± 0.011**
humidity (%)	5.73 ± 0.023	**0.69 ± 0.015**	**0.72 ± 0.016**
dew point (°F)	3.89 ± 0.027	**0.68 ± 0.012**	**0.70 ± 0.010**
wind speed (mph)	8.24 ± 0.084	**5.24 ± 0.054**	**5.20 ± 0.063**
wind gust (mph)	10.81 ± 0.073	**6.71 ± 0.057**	**6.65 ± 0.052**
pressure (Pa)	7.82 ± 0.16	**3.39 ± 0.19**	**3.42 ± 0.17**

errors on each failed sensor separately. Here too, **Refer-Neu** and **Refer-FFX** outperform **Replace** with large margins. Wind speed, wind gust and pressure are difficult to reconstruct well but **Refer-Neu** and **Refer-FFX** are significantly better than **Replace**. **Refer-FFX** performs comparably to **Refer-Neu**.

Table 3.2: Reconstruction errors (RMSE) on weather data for compound sensor failures. Each entry shows the average reconstruction error and the corresponding standard error. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Failed Sensor	**Replace**	**Refer-Neu**	**Refer-FFX**
temperature (°F)	3.94 ± 0.024	**0.69 ± 0.020**	**0.73 ± 0.018**
humidity (%)	5.73 ± 0.023	0.82 ± 0.019	**0.87 ± 0.021**
dew point (°F)	3.89 ± 0.027	0.71 ± 0.018	**0.75 ± 0.018**
wind speed (mph)	8.24 ± 0.084	**6.13 ± 0.072**	**6.07 ± 0.080**
wind gust (mph)	10.81 ± 0.073	**7.35 ± 0.073**	**7.24 ± 0.069**
pressure (Pa)	7.82 ± 0.16	**3.71 ± 0.22**	**3.82 ± 0.21**

On average, the number of parameters used in **Refer-FFX** ($\{w_i\}$ in Eq.(3.2)) is 65% less than that in **Refer-Neu** (weights in the neural networks).

Results on UUV data

Following the dataset description in Chapter 2.3, we use the concatenated sensor values in 10 trips as the source domain, and the remaining 10 trips are used as the target domain. We examine reconstruction errors on surge, heave and sway whose sensor values are crucial for higher-layer software.

Individual sensor change. We treat each of the surge, heave and sway sensors as the failed sensor, and the remaining sensors as reference sensors. Table 3.3 compares reconstruction errors of different methods, where **Refer-Neu** and **Refer-FFX** significantly outperform **Replace**. This shows that sensor relationships are helpful in reconstructing failed sensors. We present more detailed analysis in Section 6.5 where the efficacy of our adaptation is demonstrated in an end-to-end evaluation.

Table 3.3: Reconstruction errors (RMSE) on UUV data for individual sensor failures. Each entry shows the average reconstruction error and the corresponding standard error. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Failed Sensor	**Replace**	**Refer-Neu**	**Refer-FFX**
surge (m/s)	2.47 ± 0.14	$\mathbf{0.60 \pm 0.075}$	$\mathbf{0.66 \pm 0.071}$
heave (m/s)	0.13 ± 0.0068	$\mathbf{0.024 \pm 0.0051}$	$\mathbf{0.020 \pm 0.0062}$
sway (m/s)	2.31 ± 0.13	$\mathbf{0.71 \pm 0.068}$	$\mathbf{0.74 \pm 0.065}$

Compound sensor change. We treat all sensors in {surge, heave, sway} as failed sensors, and the propeller RPM and waterspeed sensors as reference sensors. Table 3.4 reports the results. Here too, **Refer-Neu** and **Refer-FFX** show clear advantages over **Replace**.

Table 3.4: Reconstruction errors (RMSE) on UUV data for compound sensor failures. Each entry shows the average reconstruction error and the corresponding standard error. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Failed Sensor	**Replace**	**Refer-Neu**	**Refer-FFX**
surge (m/s)	2.47 ± 0.14	$\mathbf{0.67 \pm 0.081}$	$\mathbf{0.71 \pm 0.084}$
heave (m/s)	0.094 ± 0.0063	$\mathbf{0.027 \pm 0.0067}$	$\mathbf{0.026 \pm 0.0073}$
sway (m/s)	2.31 ± 0.13	$\mathbf{0.75 \pm 0.072}$	$\mathbf{0.78 \pm 0.079}$

For both individual and compound sensor changes, **Refer-Neu** and **Refer-FFX** perform comparably. On average, the number of parameters used in **Refer-FFX** is 72% less than that in **Refer-Neu**.

3.2 Model-level Adaptation

Model-level adaptation attempts to adapt a model that is trained on the original sensor values. Suppose $y_1, y_2, \cdots, y_S$ are the corresponding labels before sensor failure. It is easy to see that model-level adaptation is simply the task of re-training a model using reference sensors and labels: $\{(\mathbf{x}_{s,1:K'}, y_s)\}_{s=1}^{S}$. We will present an empirical study on model-level adaptation in Chapter 5.

Chapter 4

Sensor-level Adaptation to Sensor Changes

Sensor changes often happen in real-world systems due to situations like replacement of failed sensors, sensor upgrade and energy optimization [94, 69, 62]. When sensor changes happen, a set of sensors, namely, the replaced sensors, are replaced by new sensors. When changing to new sensors, sensor values may not be consistent with old values. For example, a new sensor may measure a different type of signal compared to the sensor it replaces. Even when measuring the same type of signal, inconsistencies may still exist due to mis-calibration of the new sensors with respect to the replaced sensors. Existing work mainly focuses on detecting sensor changes but rarely addresses how to adapt to these changes [10, 53, 17, 3].

One adaptation approach is to simply ignore the new sensors and reconstruct the replaced sensors using reference sensors. This reduces to our approach in Chapter 3. However, new sensors may contain complementary information over reference sensors, which may help us better reconstruct the replaced sensors. As an extreme example, if the new sensors are exactly the same as the replaced sensors, then using their sensor values definitely aids reconstruction.

Learning a reconstruction function that exploits the new sensors poses unique challenges, since there is no overlapping period of time between the replaced and the new sensors. Classical regression methods cannot be directly applied. To address this challenge, we propose an approach called **ASC** (Adaptation to Sensor

Changes) that learns a reconstruction function to preserve sensor value distributions before and after the sensor change.

4.1 Approach

Assumptions and Intuition. Our approach reconstructs sensor values of the replaced sensors from time $S+1$ to $S+T$ based on the reference sensors and the new sensors. The underlying assumptions are:

- Sensor values from reference sensors are correlated with those from replaced sensors;
- Sensor values from reference sensors are correlated with those from new sensors.

Such assumptions typically hold in real-world systems because sensor values of different sensors are often correlated [41].

Our approach is based on the following intuition: New sensors may contain complementary information over reference sensors, useful for reconstructing replaced sensors. Fig. 4.1 illustrates this intuition, where the reference sensor, replaced sensor, and the new sensor are temperature, humidity, and dew point, respectively. The left plot shows two selected samples from historical data. We can see that for the same temperature value, humidity can take different values. The middle plot shows that if we attempt to reconstruct humidity from temperature alone, via the g function, then the reconstructed humidity values become exactly the same, since the temperature information alone is insufficient for the reconstruction. The right plot shows that by incorporating dew point as a new signal, the reconstructed humidity values are distributed similarly to those in the left plot.

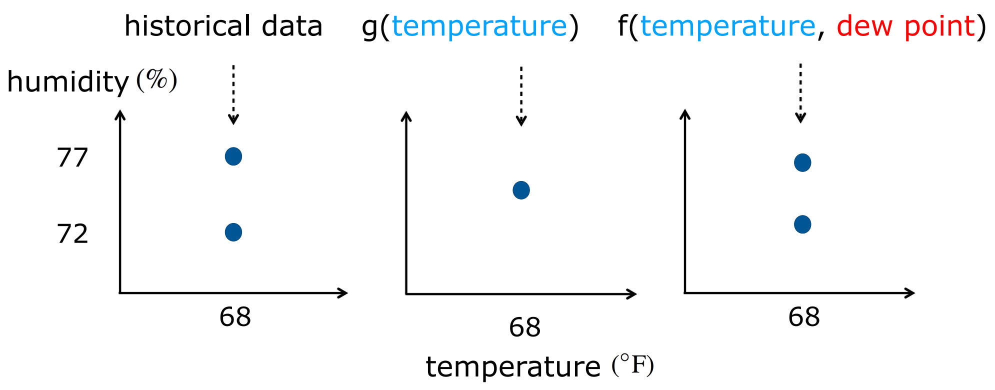


Figure 4.1: Illustration of the intuition behind **ASC**.

This is expected because dew point contains complementary information over temperature for reconstructing humidity. The above intuition leads to the key idea of our approach: to learn a reconstruction function that preserves the sensor value distributions before and after the sensor change.

Formulation. We follow the notations in Chapter 2. We refer to sensor values before the sensor change as the *source domain*, and sensor values after the sensor change as the *target domain*. Specifically, we aim to learn a reconstruction function $\mathbf{f}_{\mathbf{\Theta}}(\mathbf{z})$ that maps sensor values after the sensor change to values before the sensor change, where $\mathbf{\Theta}$ denotes the parameters of the function. Note that the output of $\mathbf{f}_{\mathbf{\Theta}}(\mathbf{z})$ is a matrix when there are more than one replaced sensor. In our implementation, we use the form

$$\mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}) = \mathbf{\Theta}^{\mathrm{T}}\mathbf{h}(\mathbf{z}) \tag{4.1}$$

where $\mathbf{h}()$ is a nonlinear feature mapping, e.g., quadratic features, or features derived from FFX [73].

We are interested in $\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z})$ such that distributions of sensor values are similar across domains after the reconstruction. This motivates us to seek $\mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z})$ such that the two sets of samples $\{\mathbf{x}_s\}$ and $\{[\mathbf{z}_{t,1:K'}; \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)]\}$ (i.e., reconstructed samples in the target domain)[1] are "mixed" as much as possible. When this happens, each source-domain sample $\mathbf{x}_s$ becomes close to its k-nearest neighbors in the target domain, and vice versa. Therefore we propose the following objective function to minimize the cross-domain k-nearest neighbor distances

$$\begin{aligned} \min_{\boldsymbol{\Theta}} \sum_{s=1}^{S} \sum_{t \in \mathcal{N}_{\mathcal{T}}^{k}(s)} & \mathcal{D}(\mathbf{x}_s, [\mathbf{z}_{t,1:K'}; \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)]) \\ + \sum_{t=1}^{T} \sum_{s \in \mathcal{N}_{\mathcal{S}}^{k}(t)} & \mathcal{D}([\mathbf{z}_{t,1:K'}; \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)], \mathbf{x}_s) + \lambda \|\boldsymbol{\Theta}\|_2^2 \end{aligned} \tag{4.2}$$

where $\mathcal{D}(\cdot,\cdot)$ is the distance function defined in the space $\mathbf{x} \in \mathcal{R}^K$. $\mathcal{N}_{\mathcal{T}}^{k}(s)$ denotes the set of indices corresponding to $\mathbf{x}_s$'s k-nearest neighbors in the target domain, and $\mathcal{N}_{\mathcal{S}}^{k}(t)$ denotes the set of indices corresponding to $[\mathbf{z}_{t,1:K'}; \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)]$'s k-nearest neighbors in the source domain. Here, nearest neighbors are determined based on the distance function $\mathcal{D}$. $\|\boldsymbol{\Theta}\|_2^2$ is the regularization term on $\boldsymbol{\Theta}$ with $\lambda \geq 0$ as the regularization parameter.

For simplicity, we set $\mathcal{D}$ to be the squared Euclidean distance[2]

$$\begin{aligned} \mathcal{D}(\mathbf{x}_s, [\mathbf{z}_{t,1:K'}; \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)]) = & \|\mathbf{x}_{s,1:K'} - \mathbf{z}_{t,1:K'}\|_2^2 \\ & + \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2 \end{aligned} \tag{4.3}$$

[1]We use the notation $1 : K'$ to denote a set of indices from 1 to K'.

[2]In our implementation, each dimension is normalized into the same scale.

Letting $v_{st}^2 = \|\mathbf{x}_{s,1:K'} - \mathbf{z}_{t,1:K'}\|_2^2$, we can write (4.2) as

$$\min_{\boldsymbol{\Theta}} \sum_{s=1}^{S} \sum_{t \in \mathcal{N}_{\mathcal{T}}^k(s)} \left(v_{st}^2 + \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2 \right) \tag{4.4}$$
$$+ \sum_{t=1}^{T} \sum_{s \in \mathcal{N}_{\mathcal{S}}^k(t)} \left(v_{st}^2 + \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2 \right) + \lambda \|\boldsymbol{\Theta}\|_2^2$$

In Eq. (4.4), $\mathcal{N}_{\mathcal{T}}^k(s)$ and $\mathcal{N}_{\mathcal{S}}^k(t)$ are dependent on $\boldsymbol{\Theta}$, making Eq. (4.4) non-smooth and non-convex in $\boldsymbol{\Theta}$.

Optimization. For the ease of optimization, we introduce a set of auxiliary variables to *decouple* the dependency of $\mathcal{N}_{\mathcal{T}}^k(s)$ and $\mathcal{N}_{\mathcal{S}}^k(t)$ on $\boldsymbol{\Theta}$. Let $\mathcal{V}_{\mathcal{T}}^k(s)$ index $\mathbf{x}_s$'s any (not necessarily the nearest) k neighbors in the target domain, and $\mathcal{V}_{\mathcal{S}}^k(t)$ index $[\mathbf{z}_{t,1:K'}; \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)]$'s any k neighbors in the source domain. It is easy to see that

$$\sum_{t \in \mathcal{N}_{\mathcal{T}}^k(s)} \left(v_{st}^2 + \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2 \right) \tag{4.5}$$
$$= \min_{\mathcal{V}_{\mathcal{T}}^k(s)} \sum_{t \in \mathcal{V}_{\mathcal{T}}^k(s)} \left(v_{st}^2 + \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2 \right) \tag{4.6}$$

and that the same relationship holds for $\mathcal{V}_{\mathcal{S}}^k(t)$ and $\mathcal{N}_{\mathcal{S}}^k(t)$. Thus (4.4) is equivalent to

$$\min_{\boldsymbol{\Theta}, \{\mathcal{V}_{\mathcal{T}}^k(s)\}, \{\mathcal{V}_{\mathcal{S}}^k(t)\}} \sum_{s=1}^{S} \sum_{t \in \mathcal{V}_{\mathcal{T}}^k(s)} \left(v_{st}^2 + \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2 \right) \tag{4.7}$$
$$+ \sum_{t=1}^{T} \sum_{s \in \mathcal{V}_{\mathcal{S}}^k(t)} \left(v_{st}^2 + \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\boldsymbol{\Theta}}(\mathbf{z}_t)\|_2^2 \right) + \lambda \|\boldsymbol{\Theta}\|_2^2.$$

(4.7) can be efficiently optimized via a procedure with two alternating steps. When $\mathbf{\Theta}$ is fixed, we update $\{\mathcal{V}_{\mathcal{T}}^k(s)\}$ and $\{\mathcal{V}_{\mathcal{S}}^k(t)\}$ based on nearest neighbor search. When $\{\mathcal{V}_{\mathcal{T}}^k(s)\}$ and $\{\mathcal{V}_{\mathcal{S}}^k(t)\}$ are fixed, we optimize $\mathbf{\Theta}$ by solving

$$\min_{\mathbf{\Theta}} \sum_{s=1}^{S} \sum_{t \in \mathcal{V}_{\mathcal{T}}^k(s)} \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}_t)\|_2^2 \tag{4.8}$$

$$+ \sum_{t=1}^{T} \sum_{s \in \mathcal{V}_{\mathcal{S}}^k(t)} \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}_t)\|_2^2 + \lambda \|\mathbf{\Theta}\|_2^2 \tag{4.9}$$

which can be easier than solving (4.4) when $\mathbf{f}_{\mathbf{\Theta}}$ is smooth in $\mathbf{\Theta}$. When $\mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}_t)$ is linear in $\mathbf{\Theta}$, the optimal $\mathbf{\Theta}$ can be computed analytically.

The above procedure decreases the value of the objective function in (4.7) in each alternating step, and converges to a local minimum of (4.4). Empirically, the procedure converges quickly (usually within 50 iterations).

Initialization. The quality of the solution depends on how we initialize $\mathbf{\Theta}$. Suppose we have a way to accurately predict the values of new sensors using $\mathbf{x}_{s,1:K'}$. Let $\mathbf{u}_s$ denote the predicted values of the new sensors. We can initialize $\mathbf{\Theta}$ by solving

$$\min_{\mathbf{\Theta}} \sum_{s} \|\mathbf{x}_{s,1:K} - \mathbf{f}_{\mathbf{\Theta}}([\mathbf{x}_{s,1:K'}; \mathbf{u}_s])\|_2^2. \tag{4.10}$$

Although estimating $\mathbf{u}_s$ can be very challenging when the correlations between the replaced sensors and the new sensors are weak, we can still estimate a candidate set for $\mathbf{u}_s$ based on target-domain data as follows: For each $\mathbf{x}_{s,1:K'}$, we find a set of its nearest neighbors in $\{\mathbf{z}_{t,1:K'}\}$, and use the corresponding $\mathbf{z}_{t,K'+1:K'+P}$ to form a candidate set $\mathcal{U}_s$. We then minimize the model error by optimizing both $\mathbf{\Theta}$ and $\{\hat{\mathbf{u}}_s\}$:

$$\min_{\mathbf{\Theta},\{\hat{\mathbf{u}}_s\}} \sum_s \min_{\hat{\mathbf{u}}_s \in \mathcal{U}_s} \|\mathbf{x}_{s,K'+1:K} - \mathbf{f}_{\mathbf{\Theta}}([\mathbf{x}_{s,1:K'}, \hat{\mathbf{u}}_s])\|_2^2 \tag{4.11}$$

where $\hat{\mathbf{u}}_s$ is allowed to be *any* element of $\mathcal{U}_s$. (4.11) essentially relaxes the dependency between the replaced sensors and the new sensors, and uses the optimal $\mathbf{\Theta}$ for the relaxed setting as an initialization. By setting $\{\mathcal{U}_s\}$ to different sizes, we can get different initial solutions for $\mathbf{\Theta}$.

Parameter Tuning. For tuning the regularization parameter λ, we use a special *leave-one-out* cross-validation strategy. We synthesize a set of sensor change scenarios by treating each sensor in the source domain as the replaced sensor, and using a biased version[3] of that sensor as the new sensor. We then select the optimal λ such that the average reconstruction error on these synthesized scenarios is minimized.

4.2 Empirical Study

We evaluate **ASC** on sensor data from the weather and UUV domains. We compare **ASC** to three baseline methods:

- **Replace**: non-adaptation method that substitutes each replaced sensor with a new sensor that has the closest mean and variance in sensor values.
- **Refer**: adaptation method that reconstructs sensor values of replaced sensors using reference sensors, without exploiting any new sensor.

[3]The biased version is created by offsetting each sensor value by the same bias.

- **ReferZ**: adaptation method that works in the following three steps:

    1. Learn a regression model on the target domain to reconstruct new sensors from reference sensors;
    2. Use the learned regression model to reconstruct new sensors on the source domain;
    3. Learn a reconstruction function on the source domain to reconstruct replaced sensors from reference sensors and reconstructed new sensors.

    This method can work well if new sensors and reference sensors are strongly correlated, which may not hold in real-world applications.

The reconstruction error of each method is measured by RMSE (Root Mean Square Error) between the reconstructed sensor values and the ground truth.

4.2.1 Results on Weather Data

Following the dataset description in Chapter 2.3, we use 30 weather stations from 10 geographical clusters. We generate random triplets across clusters. We generate each triplet in the following way: 1) randomly select two clusters; 2) randomly select two stations from the first cluster (denoted as the station A1 and A2), and one station from the second cluster (denoted as the station B). We use sensors in A1 as the compound sensor, sensors in A2 as new sensors, and sensors in B as reference sensors. We generate 100 random triplets, and report averaged results.

Each station consists of six sensors including temperature (°F), humidity (%), dew point (°F), wind speed (mph), wind gust (mph), and pressure (Pa). Sensor values are collected every 5 minutes and are temporally aligned. Sensor values from A1 and A2 are more correlated than those from A1(A2) and B. We use two years of data, with data in 2016 as the source domain and data in 2017 as the target domain.

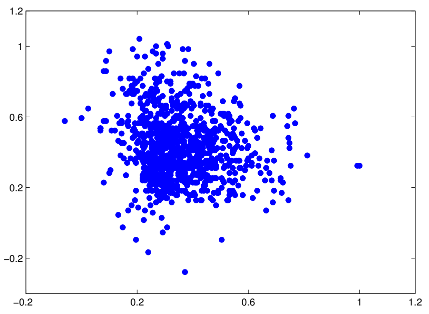
Individual sensor changes. We treat each sensor in A1 as the replaced sensor, the remaining sensors in A1 plus all sensors in B as reference sensors, and all sensors in A2 as new sensors. Table 4.1 reports average reconstruction errors and the corresponding standard errors, with Imp. showing the average improvement (in %) of **ASC** over the best baseline method. We can see that **ASC** achieves an average improvement of 6.4% over baselines. This shows high robustness of **ASC**. In general, **ASC** shows more statistically significant improvement on sensors whose values exhibit large variances (e.g., wind gust and pressure). **Replace** always underperforms compared to **Refer**, revealing that directly using new sensors can cause significant differences in sensor values. **ReferZ** performs better than **Refer** by leveraging new sensors. **ASC** further improves over **ReferZ** because it better exploits information from new sensors.

Table 4.1: Reconstruction errors (RMSE) on weather data for individual sensor changes. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of **ASC** over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

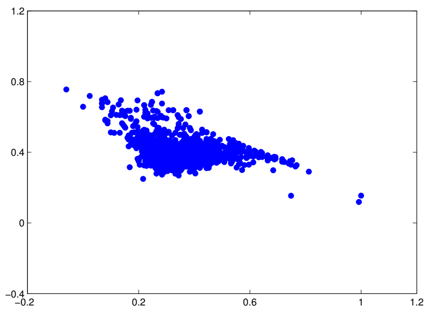
Sensor	**Replace**	**Refer**	**ReferZ**	**ASC**	Imp.
temperature	3.94 ± 0.024	0.61 ± 0.011	0.59 ± 0.009	**0.57 ± 0.010**	4.1
humidity	5.73 ± 0.023	**0.72 ± 0.016**	**0.71 ± 0.015**	**0.72 ± 0.015**	-1.7
dew point	3.89 ± 0.027	0.70 ± 0.010	**0.68 ± 0.009**	**0.67 ± 0.010**	2.8
wind speed	8.24 ± 0.084	**5.20 ± 0.063**	**5.21 ± 0.064**	**5.11 ± 0.060**	1.7
wind gust	10.81 ± 0.073	6.65 ± 0.052	6.65 ± 0.048	**6.31 ± 0.046**	5.0
pressure	7.82 ± 0.16	3.42 ± 0.19	2.48 ± 0.17	**1.83 ± 0.17**	26.2

Figure 4.2 visualizes the joint distributions over wind speed and reconstructed pressure, on a station in San Francisco (x-axis: wind speed, y-axis: reconstructed pressure). Figure 4.2 (a) is the ground-truth distribution that we would like to

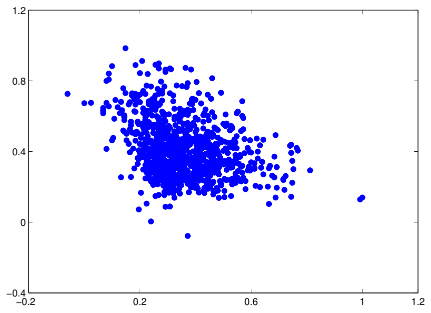
approximate after adaptation. As we can observe, **Replace** generates a significantly different joint distribution compared to the ground-truth, while **ASC** produces a much closer distribution by leveraging new sensors.



(a) ground truth



(b) **Refer**



(c) **ASC**

Figure 4.2: Visualization of wind speed and reconstructed pressure on a weather station in San Francisco (x-axis: wind speed, y-axis: reconstructed pressure produced by different approaches). Ground truth corresponds to the true pressure values. Values are in normalized scales.

Compound sensor changes. We treat all sensors in A1 as the replaced sensors, all sensors in B as reference sensors, and all sensors in A2 as new sensors. Table 4.2 reports reconstruction errors on each replaced sensor separately. **ASC** statistically outperforms baselines in three cases, achieving an average improvement of 5.7%. Compared to Table 4.1, **ASC** produces larger reconstruction errors mainly because reference sensors have lower correlations with replaced sensors in this case.

Table 4.2: Reconstruction errors (RMSE) on weather data for compound sensor changes. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of **ASC** over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	**Replace**	**Refer**	**ReferZ**	**ASC**	Imp.
temperature	3.94 ± 0.024	0.73 ± 0.018	0.71 ± 0.013	**0.68 ± 0.014**	4.2
humidity	5.73 ± 0.023	**0.87 ± 0.021**	**0.88 ± 0.020**	**0.87 ± 0.022**	0
dew point	3.89 ± 0.027	0.75 ± 0.018	**0.74 ± 0.012**	**0.72 ± 0.011**	2.6
wind speed	8.24 ± 0.084	**6.07 ± 0.080**	**6.11 ± 0.074**	**6.13 ± 0.082**	-1.8
wind gust	10.81 ± 0.073	7.24 ± 0.069	7.08 ± 0.072	**6.83 ± 0.070**	3.5
pressure	7.82 ± 0.16	3.82 ± 0.21	2.83 ± 0.20	**2.26 ± 0.18**	20.1

4.2.2 Results on UUV Data

Following the dataset description in Chapter 2.3, we use the concatenated sensor values in 10 trips as the source domain, and the remaining as the target domain. We examine reconstruction errors on surge (m/s), heave (m/s) and sway (m/s) whose sensor values are crucial for higher-layer software. To simulate new sensors, we use a biased version for the surge, heave and sway sensors. The biased version offsets the original sensor values by a sensor-specific bias. We set the bias to 3σ, where σ is the standard deviation of the original sensor values.

Individual sensor changes. We treat each of the surge, heave and sway sensors as the replaced sensor, and the remaining sensors as reference sensors. Table 4.3 compares reconstruction errors of different methods, where **ASC** improves over the best baseline by an average of 8.8%. The improvement on surge is the most statistically significant. **Refer** and **ReferZ** always outperform **Replace**, consistent with our observations on weather data.

Table 4.3: Reconstruction errors (RMSE) on UUV data for individual sensor changes. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of **ASC** over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	**Replace**	**Refer**	**ReferZ**	**ASC**	Imp.
surge	2.47 ± 0.14	0.66 ± 0.071	0.58 ± 0.048	**0.47 ± 0.051**	18.9
heave	0.13 ± 0.0068	**0.020 ± 0.0062**	**0.020 ± 0.0046**	**0.019 ± 0.0049**	6.5
sway	2.31 ± 0.13	**0.74 ± 0.065**	**0.72 ± 0.059**	**0.71 ± 0.063**	1.1

Compound sensor changes. We treat all sensors in the DVL compound sensor as the replaced sensors, and the propeller RPM and waterspeed sensors as reference sensors. Table 4.4 reports the results. **ASC** improves over the best baseline by an average of 3.0%. Compared to Table 4.3, the improvement decreases for each sensor because fewer reference sensors are used.

Table 4.4: Reconstruction errors (RMSE) on UUV data for compound sensor changes. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of **ASC** over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	**Replace**	**Refer**	**ReferZ**	**ASC**	Imp.
surge	2.47 ± 0.14	0.71 ± 0.084	0.67 ± 0.078	**0.62 ± 0.081**	6.0
heave	0.094 ± 0.0063	**0.026 ± 0.0073**	**0.026 ± 0.0070**	**0.024 ± 0.0073**	3.4
sway	2.31 ± 0.13	**0.78 ± 0.079**	**0.75 ± 0.080**	**0.75 ± 0.076**	-0.5

4.3 Evaluation in BRASS Project

In the evaluation of the BRASS Project [62] Phase 1, we conducted extensive experiments on Weather Underground Data. We organize data into 13 clusters, covering 13 regions in Los Angeles, San Francisco, Austin and Chicago. In each cluster, there are 3 weather stations, each containing 2 years of weather data. Five individual sensors (temperature, humidity, dew point, wind speed and wind gust) are used in all stations.

We evaluate our adaptation algorithms over randomly chosen clusters, stations, sensors, and time periods. Once a random cluster is chosen, we randomly pick two stations (A1 and A2). Since the two stations are from the same cluster, their sensor values are relatively similar. We further randomly pick an individual sensor from station A1, and replace it with the same individual sensor from station A2. To enable adaptation, we use training data from a 2-month time period (without sensor change), 1-month data for the adaptation period (sensor change happens in the beginning), and 1-month data for the evaluation period. The four-month data are consecutive, as shown in Fig. 4.3. The goal is to learn an adaptation function based on the data in the training and adaptation periods, and then evaluate adaptation performance in the evaluation period.

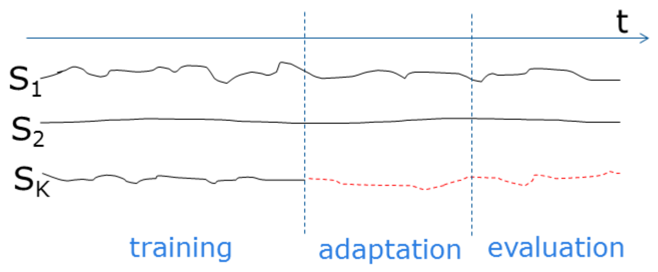


Figure 4.3: Illustration of training, adaptation and evaluation periods in the BRASS project evaluation.

To determine whether an adaptation succeeds or not, we introduce a benchmark called reference error. It defines an error bound that our system can tolerate. If adaptation error is less than reference error, we consider the adaptation successful.

In our implementation, we estimate reference error by averaging the errors between every pair of weather stations in a cluster over the evaluation period.

Table 4.5 summaries adaptation performance on random tests described above. Our evaluation is performed on cases where the error of no adaptation (i.e., direct use of the new sensor) exceeds the reference error. **ASC** achieves high success rate on temperature, humidity, dew point and wind speed. On wind gust, the success rate is relatively low, because wind gust is hard to reconstruct due to large variance. Despite some chances of failures, **ASC** shows positive improvement over reference error on all individual sensors.

Table 4.5: Adaptation performance on random tests in the BRASS project evaluation.

Sensor	Success rate (%)	Avg. Imp. over ref error (%)
temperature	95.4	61.6
humidity	96	65.8
dew point	100	71.1
wind speed	84.6	28.7
wind gust	66.7	24.0

Fig. 4.4 shows the reconstructed wind gust in one random test. The blue curve presents the wind gust from a target station and a nearby station. The red curve presents the wind gust after adaptation, which is much more similar to the original signal in the training period.

4.4 Estimating Adaptation Quality

To build survivable software, estimating the quality of adaptation is also important since it enables higher-layer software components to determine whether or not

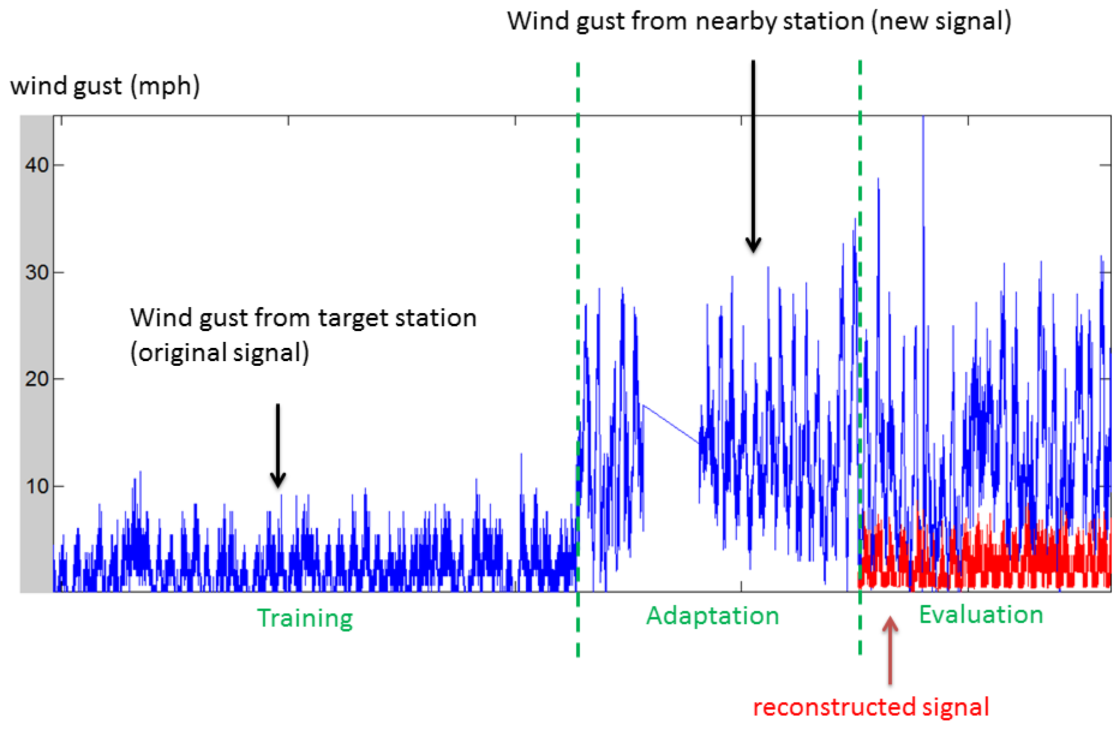


Figure 4.4: Visualization of reconstructed wind gust by **ASC**. Blue curve represents observed sensor values and red curve represents reconstructed sensor values.

to accept a proposed adaptation. Towards this end, we develop a method to estimate an error interval for the gap between the reconstructed sensor value and the ground truth.

We would like to obtain such an error interval for each reconstructed sensor value and for each sample in the target domain. Given a reconstructed sample in the target domain $[\mathbf{z}_{t,1:K'}; \mathbf{f}_{\mathbf{\Theta}}(\mathbf{z}_t)]$, we estimate its error interval for a given reconstructed sensor value from similar samples in the source domain:

1. Find its κ nearest neighbors in the source domain according to distances defined in Eq. (4.3);

2. Compute the standard deviation σ on the given reconstructed sensor value among the κ neighbors found in Step 1;

3. Set the estimated error interval to be $[-\alpha\sigma, \alpha\sigma]$, where $\alpha > 0$ is a scaling factor. An ideal α makes the error interval as *tight* as possible. α can be tuned on source-domain samples by optimizing the "excess error" notion defined below.

Excess Error of the Error Interval. To quantify the tightness of the estimated error interval, we use the notion of *excess error*. It is defined as the gap between the ground-truth value and the closest endpoint of the error interval, when the interval contains the ground-truth value. Fig. 4.5 illustrates this notion. If the interval does not contain the ground-truth value, we consider the interval invalid. In practice, we can tolerate a small failure rate of the estimated error interval by setting a recall parameter (e.g., 90%). We can then find the smallest α to achieve the given recall and compute the corresponding excess error. Clearly, we favor a smaller excess error as it results in a tighter error interval. We present the results of excess errors in the next section.

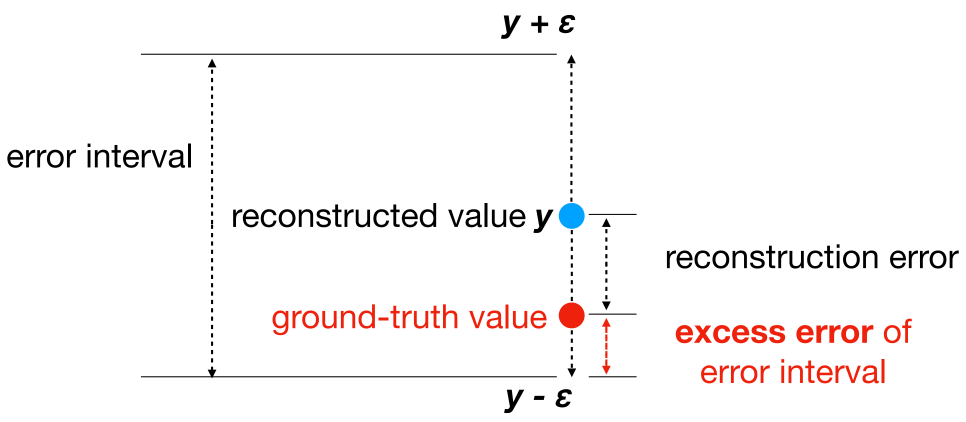


Figure 4.5: Notion of excess error of the error interval.

4.5 Ability to Exploit Many Sensors

As an increasing number of sensors are deployed in real-world systems, it is crucial for **ASC** to be able to exploit many sensors. This also enables our approach to be deployed in an open environment where new sensors continuously emerge. Dealing with a large number of sensors is challenging in two aspects:

- Noisy sensors are likely to be involved and can degrade adaptation performance. For example, if some reference sensors produce highly noisy values, the nearest neighbor distances can suffer from the noise. Also, noisy values in reference or new sensors can cause the optimization algorithm to get stuck in poor local minima;
- Large number of sensors leads to a large parameter space of $\mathbf{\Theta}$, which significantly increases the computational cost.

In addressing these issues, we develop a two-step procedure to select a subset of useful sensors:

1. Selecting a subset of reference sensors: for each reference sensor, compute the average correlation between its sensor values and those from each replaced sensor, and then select N_{ref} reference sensors with the largest average correlation scores;
2. Selecting a subset of new sensors: for each new sensor, compute the average correlation between its sensor values and those from each replaced sensor as well as each selected reference sensor in Step 1, and then select N_{new} new sensors with the largest average correlation scores.

Here, N_{ref} and N_{new} are set by the user in specific applications. We denote this improved approach as **ASC**$^{\mathsf{SEL}}$.

Experiments. We use the same triplets in Section 4.2.1. For each triplet, we use A1 as the compound sensor, and simulate reference sensors and new sensors from the remaining 29 stations. Specifically, sensors from 15 randomly selected stations are used as reference sensors, and sensors from the other 14 stations are used as new sensors. This makes the total number of sensors exceed 200.[4] Table 4.6 reports the results for individual sensor changes, where $\mathbf{ASC}^{\mathsf{SEL}}$ uses $N_{\mathsf{ref}} = N_{\mathsf{new}} = 10$. In terms of reconstruction errors, $\mathbf{ASC}^{\mathsf{SEL}}$ achieves statistically significant improvement over **ASC** in all cases. Note that $\mathbf{ASC}^{\mathsf{SEL}}$ outperforms **ASC** from Table 4.1, which reveals that a large pool of reference and new sensors actually help. In contrast, **ASC** from Table 4.6 performs worse than itself from Table 4.1 due to overfitting. This demonstrates the efficacy of our sensor selection procedure when the number of sensors is large. In terms of excess errors, $\mathbf{ASC}^{\mathsf{SEL}}$ achieves smaller values than **ASC**, consistent with the fact that $\mathbf{ASC}^{\mathsf{SEL}}$ learns better reconstruction functions. The excess errors on wind speed and wind gust are relatively large, because these sensor values exhibit large variances and are difficult to reconstruct. We observe similar trends in the scenario of compound sensor changes.

4.6 Leveraging Spatial and Temporal Information

One way to improve adaptation performance is to exploit additional information about sensors. In particular, we are interested in leveraging spatial and temporal information about sensors which can be easily obtained in practice [25, 36]. In the context of weather stations, suppose one station accesses the temperature sensor

[4]Some stations have additional types of sensors, e.g., precipitation.

Table 4.6: Individual sensor changes on weather data with many sensors. Each entry shows the average reconstruction error and the corresponding standard error. The best performing method(s) (statistically significant up to one standard error) are in bold font.

replaced sensor	Reconstruction Error		Excess Error	
	ASC	**ASC$^{\mathsf{SEL}}$**	**ASC**	**ASC$^{\mathsf{SEL}}$**
temperature (°F)	0.47 ± 0.012	**0.38 ± 0.009**	0.34 ± 0.010	**0.22 ± 0.009**
humidity (%)	0.53 ± 0.016	**0.47 ± 0.014**	0.42 ± 0.014	**0.31 ± 0.011**
dew point (°F)	0.47 ± 0.012	**0.44 ± 0.009**	0.37 ± 0.010	**0.25 ± 0.009**
wind speed (mph)	5.04 ± 0.061	**4.83 ± 0.059**	4.36 ± 0.052	**3.71 ± 0.055**
wind gust (mph)	6.28 ± 0.052	**5.61 ± 0.045**	4.75 ± 0.041	**3.96 ± 0.042**
pressure (Pa)	3.17 ± 0.19	**1.68 ± 0.18**	2.68 ± 0.19	**1.04 ± 0.18**

from another station. We can access the location information (e.g., latitude, longitude, altitude) about both sensors, as well as the exact timestamps of their sensor values. We are interested in learning calibration functions that can align sensor values from different sensors based on their spatial and temporal information. Once such calibration functions are learned, they can be used to pre-calibrate new sensors before learning adaptation functions. Intuitively, such calibration makes new sensors better aligned to old sensors and may improve the robustness and accuracy of the learned adaptation functions.

Suppose we focus on adapting sensors from station A and would like to calibrate sensor values from station B. Let x_A be a sensor value from station A, and x_B be a sensor value from station B. Note that x_A, x_B are from the same type of sensor, and x_B needs to be the closest to x_A in terms of timestamp. Let δt_B be the time difference between x_B and x_A's timestamps. For example, $\delta t_B = 2$ if x_B is received 2 time units before x_A. Additionally, let la_A, lo_A, al_A and la_B, lo_B, al_B denote the latitude, longitude and altitude of stations A and B, respectively. We can then learn a calibration function $g()$ in the following form:

$$x_A \approx g(la_A, lo_A, al_A, la_B, lo_B, al_B, x_B, \delta t_B) \tag{4.12}$$

We can further expand $g()$ to include M sensor values from station B and their time differences to x_A:

$$g(la_A, lo_A, al_A, la_B, lo_B, al_B, x_{1B}, \delta t_{1B}, x_{2B}, \delta t_{2B}, \cdots, x_{MB}, \delta t_{MB}) \tag{4.13}$$

By using more sensor values, the learned $g()$ can be more accurate and robust. We can learn $g()$ from historical data using regression methods such as neural networks. In our implementation, we use historical data covering stations from a number of different regions, so that the learned function can be highly robust. We set $M = 5$.

We apply the learned calibration function to both reference sensors and new sensors, so that their calibrated sensor values are more consistent with those values from the replaced sensors. Intuitively, this makes the overall adaptation easier, which may reduce the reconstruction error. Therefore, we can also view the overall approach as a two-step adaptation approach: the calibration as the first step, and **ASC** as the second step.

We report the adaptation results in Tables 4.7 and 4.8, where **ASC**$^{\mathsf{CALI}}$ is **ASC** applied after the calibration. **ASC**$^{\mathsf{CALI}}$ shows statistically significant improvement over **ASC** on wind speed and wind gust. These two sensors have larger variances in sensor values and the calibration effectively reduces their variances across stations before the adaptation. We also observe that the improvement on compound sensor changes is slightly larger than that on individual sensor changes.

Table 4.7: Reconstruction errors (RMSE) on weather data for individual sensor changes. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of **ASC**$^{\mathsf{CALI}}$ over **ASC**. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	**ASC**	**ASC**$^{\mathsf{CALI}}$	Imp.
temperature	0.57 ± 0.010	0.57 ± 0.009	0.3
humidity	0.72 ± 0.015	0.71 ± 0.015	1.4
dew point	0.67 ± 0.010	0.67 ± 0.009	0.0
wind speed	5.11 ± 0.060	**4.98 ± 0.061**	2.6
wind gust	6.31 ± 0.046	**6.18 ± 0.050**	2.1
pressure	1.83 ± 0.17	1.80 ± 0.15	1.5

Table 4.8: Reconstruction errors (RMSE) on weather data for compound sensor changes. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of **ASC**$^{\mathsf{CALI}}$ over **ASC**. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	**ASC**	**ASC**$^{\mathsf{CALI}}$	Imp.
temperature	0.68 ± 0.014	0.67 ± 0.012	0.9
humidity	0.87 ± 0.022	0.86 ± 0.018	1.5
dew point	0.72 ± 0.011	0.72 ± 0.010	0.2
wind speed	6.13 ± 0.082	**5.94 ± 0.083**	3.2
wind gust	6.83 ± 0.070	**6.65 ± 0.068**	2.7
pressure	2.26 ± 0.18	2.23 ± 0.19	1.3

Chapter 5

Model-level Adaptation to Sensor Changes

5.1 Problem Setting

Model-level adaptation attempts to adapt a model trained on the source domain to the target domain. In this chapter, we follow the same problem setting in Chapter 2, except that we further assume the availability of class labels for source-domain samples. We denote y_s as the label of $\mathbf{x}_s, s = 1, \cdots, S$. Note that there is no label in the target domain.

The above problem setting is typically referred to as *unsupervised domain adaptation*. It is especially challenging as the target domain does not explicitly provide any information on how to optimize classifiers.

5.2 Approach

Most of the existing approaches [89, 14, 61, 78, 49] follow a two-stage learning paradigm. They first identify a domain-invariant feature space such that the marginal distributions of the two domains are the same in the new feature space. Then, these approaches learn classifiers in the new space and expect the learned classifiers to perform equally well in both domains. Theoretical analyses have showed that the loss on the target domain for any labeling function depends on the difference between the marginal distributions, thus justifying the need to identify a feature space such that the two domains look alike [13, 71].

We hypothesize that this view and practice of two-stage learning are restrictive. One possible fallacy is that maximizing the similarity in marginal distributions bears no direct consequence on (dis)similarities between posterior distributions. Thus, if there are multiple feature spaces where the source and the target domains have similar marginals, there is no reason to believe that a classifier trained on an arbitrarily chosen one would necessarily perform well on the target domain. As an extreme case, projecting features onto irrelevant feature dimensions would make the two domains look very much alike!

Hence, the caveat is to retain *discriminative* information for constructing classifiers while we search for the domain-invariant feature space. This seems relatively straightforward to achieve if all we care is the discriminative information about the labels in the source domain. However, our main goal is to have good classifiers for the target domain. Thus, our challenge is about ***how to be discriminative without labels***.

To address this challenge, we propose a novel learning algorithm for unsupervised domain adaptation as an extension of our previous work [88], which is also described in this chapter. As opposed to the existing two-stage approaches where new feature spaces and classifiers are separately optimized, our approach combines the two in a single stage. Moreover, the new feature space is discriminative with respect to the target domain.

Main Idea. We assume that *discriminative clustering* is possible. In other words, we assume that data in both the source and target domains are tightly clustered and clusters correspond to classes. We also assume that for the same class, the clusters from the two domains are geometrically close to each other. Fig.

5.1 illustrates these two assumptions and how they can be exploited for adaptation. Leveraging these assumptions, our formulation of learning the optimal feature space balances two forces: maximizing domain similarity that makes the source and target domains look alike, and (approximately) minimizing expected classification error on the target domain. We define these two forces with information-theoretic quantities: the domain similarity being the negated mutual information between all data and their binary domain labels (SOURCE versus TARGET) and the expected classification error being the negated mutual information between the target data and its cluster (i.e., class) labels estimated from the source data. These two quantities are directly motivated by the nearest neighbor classifiers we use in the new feature space.

Our adaptation approach can be applied to both scenarios of individual sensor changes and compound sensor changes. It jointly learns two transformation matrices, one for the source domain and one for the target domain. In the special case that both domains have the same number of individual sensors, our approach learns one common transformation matrix.

Our objective is to construct a target-domain classifier $f : \mathbf{z} \in \mathcal{R}^{K'+P} \rightarrow y$. We would like the classifier to perform well on the target domain from which $\mathbf{z}_t$ is sampled. This is inherently an ill-posed problem as we do not have any labels from the target domain.

To overcome this difficulty, we leverage the *discriminative clustering* assumptions which previously described. We assume that there is a latent feature space such that i) data in the source and target domains form well-separated clusters and the clusters correspond to labels; and ii) the clusters from the source domain are geometrically close to those from the target domain if they have the same labels.

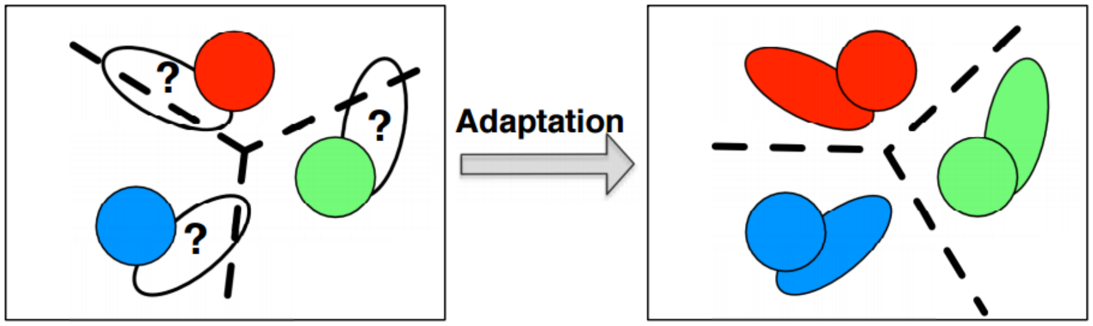


Figure 5.1: Schematic illustration of our main idea on exploiting discriminative clustering for unsupervised domain adaptation. Data in the source domain (within circles) and the target domain (within ovals) are tightly clustered, corresponding to their class labels. Moreover, clusters from the two domains are "aligned" if they correspond to the same class. Assuming and exploiting such structures in the data, classifier boundaries for the source domain (dashed lines in the left diagram) are adapted discriminatively to the target domain (dashed lines in the right diagram), minimizing the expected classification errors on the target domain. The target data is then classified with adapted classifiers.

We show how these assumptions can be used to derive information-theoretic quantities which reflect data characteristics in each domain. These quantities are parameterized in terms of the latent feature space which is in turn a linear transformation of the original feature space. We then show how to combine these quantities so that the optimal linear transformations can be learned from data. We begin by describing a few key notions.

Conditional Models In the Feature Space. Let the dimensionality of the latent feature space be d. Consider the latent feature space induced by a linear transformation $\mathbf{L} \in \mathcal{R}^{d\times K}$ on $\mathbf{x}$ and a linear transformation $\mathbf{B} \in \mathcal{R}^{d\times(K'+P)}$ on $\mathbf{z}$. In the new feature space, we use k-nearest neighbors (kNN) for classification since we assume that data form well-separated clusters. Moreover, we choose $k = 1$ to avoid cross-validating this parameter.

Let $\mathbf{u}$ be a point in the latent feature space. Let $\mathbf{u}_s = \mathbf{L}\mathbf{x}_s$ and $\mathbf{u}_t = \mathbf{B}\mathbf{z}_t$. The squared distance between two points $\mathbf{u}_i$ and $\mathbf{u}_j$ in this feature space is thus given by $d_{ij}^2 = \|\mathbf{u}_i - \mathbf{u}_j\|_2^2$.

Given a point $\mathbf{u}_i$ and a set of data points $\{\mathbf{u}_j\}$ that do not contain $\mathbf{u}_i$, we use the following model

$$p_{ij} = \frac{e^{-d_{ij}^2}}{\sum_j e^{-d_{ij}^2}} \tag{5.1}$$

to define the conditional probability of having $\mathbf{u}_j$ as $\mathbf{u}_i$'s nearest neighbor.

The above conditional model has been used in many contexts, including metric learning [46], dimensionality reduction [59], etc. Characterizing how close a point $\mathbf{u}_i$ is to other points, this model gives rise to an estimate of the posterior $p(y_i = c|\mathbf{u}_i)$ for labeling $\mathbf{u}_i$ with the class label c, assuming the class labels of $\{\mathbf{u}_j\}$ are known,

$$\hat{p}_{ic} = \sum_{j \neq i} p_{ij}\delta_{jc} \tag{5.2}$$

where δ_{jc} is 1 if $\mathbf{u}_j$'s label is c, and 0 otherwise. Since p_{ij} is a normalized probability, $\hat{p}_{ic}$ is normalized as well. For example, if the label of $\mathbf{u}_i$ is known, $\sum_c \hat{p}_{ic}\delta_{ic}$ would be the probability of correctly classifying $\mathbf{u}_i$.

Discriminative Clustering in the Source. To derive a classifier that can perform well on the target domain, we would certainly need the classifier to perform well on the source domain because we assume that the two domains share similar clustering structures. Thus, our first desideratum is to minimize the expected classification error on the source domain, when we classify it using 1-NN. This error is estimated using 1 minus the empirical average of the leave-one-out accuracy for any given point $\mathbf{u}_s$ in the source domain:

$$\varepsilon_s = 1 - \frac{1}{\mathsf{N}} \sum_s \sum_c \hat{p}_{sc} \delta_{sc} \tag{5.3}$$

Note that, if we minimize this error alone and ignore the target domain, we arrive at the metric learning technique in [46].

Discriminative clustering in the target. Since we do not have labels on the target domain, we cannot define the expected classification error as we did in Eq. (5.3) for the source domain. The challenge, therefore, is ***how to be discriminative without using labels***.

Consider an instance $\mathbf{u}_t$ from the target domain and all the instances $\{\mathbf{u}_s\}$ from the source domain. The conditional model p_{ts} of Eq. (5.1) gives rise to the probability of having a particular $\mathbf{u}_s$ as the nearest neighbor of $\mathbf{u}_t$. Using this conditional model in conjunction with the source labels to compute the posterior as in Eq. (5.2) would be incorrect for the target domain. However, if our assumptions about the two sets of clusters being geometrically close to each other indeed hold in the dataset, then the estimate $\hat{p}_{tc}$ should be close to the true posterior.

If $\hat{p}_{tc}$ approximates the true posterior well and our assumption that the target data are well clustered holds, then we can reasonably expect the C-dimensional probability vector $\hat{\mathbf{p}}_t = [\hat{p}_{t1}, \hat{p}_{t2}, \ldots, \hat{p}_{t\mathsf{C}}]$ to look like an *ideal* posterior probability vector $[0, 0, \ldots, 1, \ldots, 0]$ where the only nonzero element 1 occurs at the position corresponding to the correct label.

Since we do not know the true label, we cannot directly measure the similarity of $\hat{\mathbf{p}}_t$ to the correct and ideal posterior vector. Nonetheless, we can express our desideratum as reducing the entropy of $\hat{\mathbf{p}}_t$ such that it contains the least amount of confusing labels.

Let $H[\mathbf{p}]$ denote the entropy of a probability vector $\mathbf{p}$. If we minimize $\sum_t H[\hat{\mathbf{p}}_t]$ alone, we could arrive at a degenerate solution where every point $\mathbf{x}_t$ is assigned to the same class. To avoid this, we instead maximize the mutual information between the projected data U in the latent feature space and the estimated label $\hat{Y}$ using $\hat{\mathbf{p}}$,

$$I_t(U;\hat{Y}) = H[\hat{\mathbf{p}}_0] - \frac{1}{T}\sum_t H[\hat{\mathbf{p}}_t] \tag{5.4}$$

and the prior distribution $\hat{\mathbf{p}}_0$ is given by $\hat{\mathbf{p}}_0 = 1/T(\sum_t \hat{\mathbf{p}}_t)$. Note that using the empirical distribution of the labels in the source domain to estimate the prior $\hat{\mathbf{p}}_0$ could still lead to degenerate solutions when the labels are uniformly distributed.

Minimizing the entropy (or similarly, maximizing the mutual information) has been previously studied in the context of (discriminative) clustering [47, 38]. This criterion identifies a feature representation that classifiers can use to achieve a lower-bound of misclassification error, due to Fano's inequality [42].

Discriminability: source versus target domains. The previous discussion on discriminative clustering in the target domain hinges on the assumption that clusters for the source and the target domains are not too far from each other. We quantify this notion more precisely in the following paragraphs. Conceptually, it is similar to the idea in existing work that makes marginal distributions similar across domains.

Why is such a notion desirable? In order to use the source domain's labels as a proxy to estimate the posterior probabilities for the target data (as in Eq. (5.2)), we would like the source and the target domains to share some common probability supports in the feature space. In particular, consider the case where we classify two instances $\mathbf{u}_t$ and $\mathbf{u}_{t'}$ from the target domain. They are deemed to have the same label c if there are plenty of labeled source data in class c in their neighborhoods. We would then expect that, with high likelihood, $\mathbf{u}_t$ and $\mathbf{u}_{t'}$ are in each other's set of nearest neighbors too; otherwise, the cluster corresponding to class c in the target domain would not be very "tight".

Having instances from both domains in $\mathbf{u}_t$'s set of nearest neighbors thus entails the following. If we create a binary classification problem and assign $q_i = 1$ when $\mathbf{u}_i$ is from the source domain and $q_i = 0$ when $\mathbf{u}_i$ is from the target domain, then given $\mathbf{u}_i$, we are unable to determine well above chance level where this instance comes from.

Instead of constructing an actual binary classifier, we express our desideratum as minimizing the mutual information between the data sample U in the latent feature space and its (binary) domain label Q. Analogous to Eq. (5.4), the mutual information is given by,

$$I_{st}(U; Q) = H[\hat{\mathbf{q}}_0] - \frac{1}{S+T} \sum_i H[\hat{\mathbf{q}}_i] \tag{5.5}$$

where $\hat{\mathbf{q}}_i$ is the two-dimensional posterior probability vector of assigning $\mathbf{u}_i$ to either the source or the target domains, given *all* other data points from the two domains. Concretely, the probability is computed according to Eq. (5.2), except for the class label δ_{jc} being replaced by the domain label of $\mathbf{u}_j$. The estimated prior distribution $\hat{\mathbf{q}}_0$ is computed as $1/(S+T)(\sum_i \hat{\mathbf{q}}_i)$.

One might wonder why we do not compute and minimize the expected error as in the source domain classification Eq. (5.3). This is because we would like to leave some room for the possibility that a certain portion of the data in one domain could be "outliers" to the other domain. Minimizing domain classification error would have the adverse effect of forcing the two domains to be exactly the same. For instance, a degenerate solution would be to map every point to the origin of the feature space.

We mention in passing that the accuracy of a binary domain classifier reflects similarities between domains [15], thus approximating the original intractable combinatorial measure of similarities [13].

Learning and model selection. We have described three information-theoretic quantities: classification accuracies on the source domain ε_S of Eq. (5.3), discriminative clustering on the target domain $I_t(U; \hat{Y})$ of Eq. (5.4), and discriminability between the source and the target domains $I_{st}(U; Q)$ of Eq. (5.5).

These quantities have been derived from our assumptions about the source and target domains, specifically, the discriminative clustering structures. They are all parameterized in the linear transformations $\mathbf{L}$ and $\mathbf{B}$.

We learn the optimal $\mathbf{L}$ and $\mathbf{B}$ by balancing these quantities in the following optimization problem

$$\begin{aligned} &\textsf{minimize } -I_t(U; \hat{Y}) + \lambda I_{st}(U; Q) \\ &\textsf{subject to } \textsf{Trace}(\mathbf{L}^{\mathrm{T}}\mathbf{L}) \leq K, \ \textsf{Trace}(\mathbf{B}^{\mathrm{T}}\mathbf{B}) \leq K' + P \end{aligned} \tag{5.6}$$

where the constraints are used to control the scale of distances computed using $\mathbf{L}$ and $\mathbf{B}$.

The regularization coefficient λ needs to be cross-validated. We choose the optimal λ that attains the minimum value of ε_S. Intuitively, ε_S is defined on the source domain with labeled data and is therefore more sensible to be used for model selection. Other ways of combining these quantities were also experimented with, although the above performs the best in practice.

We comment briefly on the difference between our formulation and the entropy minimization framework for semi-supervised learning [50]. Their goal is to reduce uncertainty of labeling the unlabeled data. Thus, they use only the entropy term Eq. (5.2). More distinctively, they do not need to make the two domains look alike and thus there is no need for them to learn a feature space, nor to include a term to minimize the discriminability between the domains.

Numerical Optimization. Eq. (5.6) is a non-convex optimization problem. We use gradient-based methods to optimize the objective function. We use the PCA of the source-domain data to initialize $\mathbf{L}$ and the PCA of the target-domain data to initialize $\mathbf{B}$.

5.3 Empirical Study

In this section, we first evaluate our adaptation method on object recognition and sentiment analysis tasks. For these two tasks, the source and target domains share the same feature space; thus we have $\mathbf{L} = \mathbf{B}$, and only optimize one matrix. These experimental results are in our earlier publication [88]. We later evaluate our method in the context of model-level adaptation to sensor changes, where the number of individual sensors in the source and target domains can be different. In this case, we learn both $\mathbf{L}$ and $\mathbf{B}$ jointly.

5.3.1 Object Recognition and Sentiment Analysis

Object recognition. We use four databases of object images: Caltech-256 [51], Amazon (images from online merchants's catalogues), Webcam (low-resolution images by web cameras), and DSLR (high-resolution images by digital

SLR cameras). The last three datasets were studied in [49, 85]. Caltech-256 is added to increase the diversity of the domains.

We treat each dataset as a domain. There are 10 common object categories: backpack, coffee-mug, calculator, computer-keyboard, computer-monitor, computer-mouse, head-phones, laptop-101, touring-bike, and video-projector. There are 2533 images in total, with 8 to 151 images per category per domain.

Following the experimental protocols in previous work [85], we extract SURF features [11] and encode each image with a 800-bin histogram (the codebook is trained from a subset of Amazon images). The histograms are first normalized to have zero mean and unit standard deviation in each dimension.

For each pair of source and target domains, we conduct experiments in 20 random trials. In each trial, we randomly sample labeled data in the source domain as the training set, and unlabeled data in the target domain as the testing set.

Sentiment analysis. We use the dataset that consists of Amazon product reviews on four product types: kitchen appliances, DVDs, books and electronics [15]. Each product type is used as a separate domain. Each domain has 1,000 positive and 1,000 negative reviews. To reduce computational cost, we select the top 400 words of the largest mutual information with the labels. We then represent each review with a 400-dimensional vector of term counts (ie, bag-of-words). The vectors are normalized to have zero mean and unit standard deviation in each dimension.

For each pair of source and target domains, we conduct experiments in 10 random trials. In each trial, we randomly sample 1,600 labeled data in the source domain as the training set, and all data in the target domain as the testing set.

Classification. We learn the feature transformation $\mathbf{L}$ by solving the optimization problem Eq. (5.6). We then transform all the data using the matrix and apply 1-nearest neighbor (1-NN) to classify instances from the target domain. 1-NN is used to avoid tuning the number of nearest neighbors.

Hyperparameter tuning. Our method has two hyper-parameters: the dimensionality of the latent feature space and the regularization coefficient λ in Eq. (5.6). We cross-validate them using the model selection procedure described in Section 5.2. The range of search for the dimensionality is $\{20, 40, 70, 100\}$ and for λ is $\{0, 0.25, 1, 4, 16, 64\}$. For baselines we compare to, we follow their procedures for tuning hyper-parameters.

We compare extensively to several methods.

- **PCA**, where we project all data into the PCA directions computed on the target domain.

- **LMNN** [99], where we train a large margin nearest neighbor classifier using only the source-domain labeled data.

- Transfer Component Analysis (**TCA**) [78]. This method finds a low-dimensional linear projection such that the source and the target domains have similar marginal distributions, regularized by preserving variances in all the data. To measure similarities in marginals, the method maps data to a kernel feature space. We use Gaussian RBF kernels.

- Geodesic Flow Subspaces (**GFS**) [49]. This method interpolates (on Grassman manifold) between the PCA subspaces computed on the source and the target domains respectively. The interpolated subspaces are then used to transform the original features to form super-vectors. The dimensionality of the super-vectors is then reduced before applying 1-NN for classification.

- Structural Correspondence Learning (**SCL**) [16]. This method augments original features with linearly transformed features. The linear transformation is computed as the principal directions of parameters in binary classifiers predicting whether pivot features are present or not. In our experiments, we have used all 400 features as pivot features. We then train SVMs with the augmented feature vectors on the source domains and apply the resulting classifiers to the target domains.

Table 5.1 and Table 5.2 summarize the classification accuracies as well as standard errors of all the above methods, including **Ours** (we did not apply **SCL** to object recognition as it is difficult to define what pivot features are for those types of data). We chose a subset of all pairs for reducing experimentation time. The best performing algorithm(s) (statistically significant up to one standard error) for each pair are in bold font.

In Table 5.1 on object recognition, **Ours** performs the best on 5 out of 6 pairs, outperforming other competing methods by a large margin. On the DSLR-Amazon pair, **Ours** performs worse than **LMNN**, but still significantly better than others.

Of particular interest is that **LMNN** outperforms other methods specifically designed for domain adaptation (excluding **Ours**). This confirms our hypothesis: the two-stage learning schemes adopted by **TCA** and **GFS** suffer from the fallacy that maximizing marginal similarity does not necessarily lead to well-performing classifiers on the target domain. In particular, we believe that such methods could actually destroy discriminative information by forcing the domains to be similar.

The results thus support our argument that one-stage learning, namely identifying jointly discriminative clustering and low-dimensional feature spaces, is crucial for domain adaptation.

Table 5.1: Classification accuracies on target domains (object recognition task). Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of **Ours** over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Source → Target	**PCA**	**TCA**	**GFS**	**LMNN**	**Ours**
DSLR → Webcam	80.6±0.5	66.2±0.5	75.5±0.4	81.3±0.4	**83.6±0.5**
DSLR → Amazon	35.1±0.3	31.4±0.2	35.7±0.5	**42.3±0.3**	39.6±0.4
Caltech → DSLR	36.6±1.2	33.1±0.8	36.5±0.9	37.2±1.1	**44.4±1.2**
Caltech → Amazon	37.7±0.5	34.9±0.4	37.9±0.5	43.2±0.4	**49.2±0.6**
Amazon → Webcam	33.1±0.6	26.5±0.8	32.8±0.7	35.2±0.8	**38.5±1.3**
Amazon → Caltech	35.9±0.3	29.3±0.3	36.1±0.5	37.6±0.4	**40.0±0.4**

Table 5.2: Classification accuracies on target domains (sentiment analysis task) Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of **Ours** over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Source → Target	**PCA**	**SCL**	**TCA**	**GFS**	**LMNN**	**Ours**
Kitchen → DVD	66.1±0.7	73.2±0.6	64.9±0.5	67.9±1.0	70.8±0.5	**75.4±0.6**
DVD → Books	66.4±0.4	**79.2±0.4**	64±0.7	70.8±0.6	71.7±0.6	78.4±0.5
Books → Elec.	63.6±0.9	75.6±0.6	62.7±0.7	67.2±1.0	69.2±0.6	**79.2±0.9**
Elec. → Kitchen	71.8±0.4	**84.5±0.5**	69.5±0.7	75.8±1.2	77.3±0.6	82.9±0.5

The results on sentiment analysis in Table 5.2 also strongly support similar conclusions. Note that both **SCL** and our methods outperform other methods significantly. Our methods perform better on 2 out of 4 pairs, though slightly worse than **SCL** on the other two.

5.3.2 Weather Condition Classification

We evaluate our model-level approach on a weather condition classification task based on Weather Underground data. We use the weather condition as the class label, which is one of three possibilities: cloudy, clear and rainy. We consider six pairs of source and target domains (LA → SF, SF → LA, LA → AU, AU → LA, SF

$\rightarrow$ AU, AU $\rightarrow$ SF). For each pair, we conduct experiments in 10 random trials. In each trial, we randomly select 3,000 samples in the source domain as the training set and 3,000 samples in the target domain as the test set.

Classification. We learn the feature transformations $\mathbf{L}$ and $\mathbf{B}$ by solving the optimization problem Eq. (5.6). We then transform all the data using the learned matrices and apply 1-nearest neighbor (1-NN) to classify instances from the target domain.

Hyperparameter tuning. We set the range of search for the dimensionality as $\{4, 5, 6\}$ and for λ as $\{0, 0.25, 1, 4, 16, 64\}$.

We compare our method to baseline methods **PCA**, **TCA** and **GFS** described above. We report the results in Table 5.3. On 5 out of 6 pairs, **Ours** improves over the best baseline method. **Ours** performs comparably to **TCA** and **GFS** only on the pair SF $\rightarrow$ AU. The results demonstrate the efficacy of our approach for model-level adaptation.

Table 5.3: Classification accuracies on target domains with model-level adaptation. Each entry shows the average reconstruction error and the corresponding standard error. Imp. shows the average improvement (in %) of **Ours** over the best baseline method. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Source $\rightarrow$ Target	**PCA**	**TCA**	**GFS**	**Ours**	Imp.
LA $\rightarrow$ SF	81.2 $\pm$ 0.5	78.3 $\pm$ 0.6	80.7 $\pm$ 0.5	**84.6 $\pm$ 0.6**	4.2
SF $\rightarrow$ LA	81.0 $\pm$ 0.5	80.6 $\pm$ 0.5	81.4 $\pm$ 0.4	**85.2 $\pm$ 0.4**	4.7
LA $\rightarrow$ AU	69.4 $\pm$ 0.7	68.5 $\pm$ 0.8	70.5 $\pm$ 0.6	**72.4 $\pm$ 0.7**	2.8
AU $\rightarrow$ LA	70.3 $\pm$ 0.4	71.2 $\pm$ 0.4	71.6 $\pm$ 0.5	**74.3 $\pm$ 0.4**	3.8
SF $\rightarrow$ AU	73.7 $\pm$ 0.6	**74.8 $\pm$ 0.6**	**75.9 $\pm$ 0.5**	**75.8 $\pm$ 0.6**	-0.2
AU $\rightarrow$ SF	72.1 $\pm$ 0.3	72.4 $\pm$ 0.4	73.2 $\pm$ 0.4	**75.1 $\pm$ 0.3**	2.6

Chapter 6

Joint Detection and Adaptation to Sensor Failures

6.1 Overview

In Chapters 3 and 4, we assumed that failures are known and focused on sensor-level adaptation. In practice, however, sensor failures are often unknown. In this chapter, we present a novel machine learning framework called **JDA** (Joint Detection and Adaptation) that performs sensor failure detection and adaptation *jointly*.

Similar to our sensor-level adaptation approach in Chapter 3, the key of **JDA** is to exploit the *reconstruction relationships* among sensors, i.e., how one sensor value can be reconstructed from other sensor values. This is based on the observation that, in real-world systems, sensor values are often correlated [41]. Taking weather sensors as an example, temperature, dew point and humidity are highly correlated [2]; and each sensor value can be efficiently reconstructed from the other two. While reconstruction relationships can be generally complex, our framework decomposes this complexity into a set of simpler constraints. In particular, it uses a substrate of inequality constraints that resemble

$$(\text{temperature} - f(\text{dew point}, \text{humidity}))^2 \leq \epsilon^2, \tag{6.1}$$

where $f()$ is a function that captures known sensor relationships, and ϵ^2 is the corresponding error bound. These constraints provide a joint view for sensor failure detection and adaptation when new sensor value readings come in.

- Detection: Our framework checks each constraint, and a sensor failure is reported if one or more constraints are violated. We then infer the likely failed sensor(s) from the violated constraints.

- Adaptation: Once the failed sensors are identified, our framework reconstructs the failed sensor values from the remaining working sensor values by solving the set of constraints. Tighter constraints correspond to more accurate reconstruction relationships.

By using the same set of constraints for both detection and adaptation, our approach provides an extensible way to address the interrelated problems in one unified framework.

One important challenge in our framework is that the functions $f()$ are not necessarily given to us beforehand. Thus, a second operating idea in our framework is to extract them from historical sensor data. The extraction procedure considers different combinations of sensors and derives the functions $f()$ using nonlinear regression methods [73]. Compared to existing detection methods that extract only linear relationships [36], our extraction procedure not only enables learning more complex functions $f()$ but also results in lower reconstruction errors produced by the entire framework.

To enhance the usefulness of the proposed framework for practical applications, we provide one additional feature: when a sensor failure occurs, we not only detect it but also identify its mode of failure. This enables our detection procedure to provide additional information to higher layers of the software system, which in turn facilitates faster recovery operations. We extract features from both observed and reconstructed sensor values within a time window and classify them into five common modes of failure (Outlier, Spike, Stuck-at, High-noise and Miscalibration) [76].

An empirical study of sensor data from the weather, appliances energy and UUV domains shows that our framework detects sensor failures more accurately than other competing methods. The results also demonstrate the overall efficacy of our constraint-based framework in: (a) successfully identifying different modes of sensor failures, (b) adapting to failures by efficiently reconstructing the required sensor values, and (c) estimating the qualities of the reconstructed sensor values for higher-level decisions.

6.2 Approach

Our framework exploits the observation that real-world systems are often equipped with sensors that are correlated with each other. Such correlations could exist either between different sensor types (e.g., temperature, dew point and humidity from the same weather station) or within the same sensor type (e.g., wind speed in nearby weather stations). In this chapter, we explore a specific type of relationship between sensor values that can be characterized by a *reconstruction function* $f()$. A simple example illustrates this concept. Consider humidity HU in %. It is well known that it can be accurately determined by temperature TP in °C and dew point DP in °C [2]:

$$HU \approx f(TP, DP) = 100 \exp(\frac{aDP}{b + DP} - \frac{aTP}{b + TP}). \tag{6.2}$$

Here, f serves as a reconstruction function that takes input sensor values TP and DP and outputs sensor value HU. a and b are constants. In practice, the following constraint holds between the different sensor values.

$$(HU - f(TP, DP))^2 \leq \epsilon^2, \tag{6.3}$$

where ϵ^2 is an error bound that intrinsically measures the reconstruction quality of HU via $f(TP, DP)$. In addition, ϵ^2 can be derived from historical sensor data. For instance, we can set ϵ^2 to be the minimum value such that 95% of historical sensor values satisfy Eq. (6.3).

Assuming that the sensors for TP and DP work correctly, a failure in the sensor for HU is characterized by the violation of the constraint in Eq. (6.3). In fact, in such a case, we can even adapt to the failure by reconstructing HU via $f(TP, DP)$; and doing so automatically satisfies Eq. (6.3). Additionally, ϵ^2 in Eq. (6.3) provides an estimate of the adaptation quality (discussed in Section 4.4). However, the general challenge is that the sensors for TP and DP may not always work correctly either. If one or more of them fail in addition to the failure of the sensor for HU, Eq. (6.3) can neither be used to detect this failure nor can it be used to reconstruct the value of HU. This problem persists whether or not $f()$ is explicitly known and whether or not it is learned using state-of-the-art machine learning methods. Our framework therefore uses an additional layer of reasoning beyond just a direct application of machine learning methods to learn relationships between sensor values. In particular, it builds a substrate of constraints that retain enough simplicity individually and yet capture enough complexity and redundancy collectively. Our constraint-based framework can therefore be effectively used to first address the problem of sensor failure detection and then address the problem of failed sensor value reconstruction.

We assume that any sensor value can be accessed at any time. In the first step that addresses sensor failure detection, we are interested in detecting the possible failure of sensor k at a desired time t, i.e., determining whether or not $x_{t,k}$ should be deemed as being reliable. Of course, doing so allows us to detect sensor failures instantly, without having to wait for a time window of sensor values. We also assume that we are given N inequality constraints with reconstruction functions. (We discuss how to actually derive such reconstruction functions in Section 6.2.3.) Each such inequality constraint describes the relationship between a set of input sensor values and an output sensor value. Specifically, the nth constraint is as follows.

$$(y_n - f_n(\boldsymbol{\gamma}_n))^2 \leq \epsilon_n^2, \tag{6.4}$$

- y_n: output sensor value at some time t, e.g., $y_n = \mathbf{x}_{1,t}$;
- $\boldsymbol{\gamma}_n$: input sensor values at time $\leq t$, e.g., $\boldsymbol{\gamma}_n = [\mathbf{x}_{2,t}, \mathbf{x}_{3,t}]$. Note that $\boldsymbol{\gamma}_n$ can also involve input sensor values at time $\leq t$, e.g., $\boldsymbol{\gamma}_n = [\mathbf{x}_{2,t}, \mathbf{x}_{3,t}, \mathbf{x}_{1,t-1}, \mathbf{x}_{2,t-1}]$, where $\mathbf{x}_{1,t-1}$ and $\mathbf{x}_{2,t-1}$ can be treated as additional input sensor values;
- $f_n()$: reconstruction function that attempts to reconstruct y_n from $\boldsymbol{\gamma}_n$ derived from historical sensor data;
- ϵ_n^2: a reconstruction error bound derived from historical sensor data.

6.2.1 Detecting Sensor Failures

As the system receives sensor readings, it can check each constraint and identify the violated ones at any given time t. If the nth constraint is violated, then at least one sensor involved in that constraint has likely failed. Furthermore, the system can infer the set of failed sensors from the set of violated constraints. To do this, we first introduce K Boolean variables $\{v_k\}$, for $k = 1, 2, \cdots, K$, where v_k is 1 if sensor k has failed, and is 0 otherwise. The existence of at least one failed sensor corresponding to each violated constraint translates to a set of linear constraints on $\{v_k\}$. For instance, if a violated constraint involves sensor 1 and sensor 3, then the corresponding linear constraint is $v_1 + v_3 \geq 1$, since at least one of v_1, v_3 should have value 1. More generally, if the nth constraint is violated, then the sum of all v_k involved in $[\boldsymbol{\gamma}_n, y_n]$ should be greater than or equal to 1.

$$\sum_{k \in [\boldsymbol{\gamma}_n, y_n]} v_k \geq 1 \tag{6.5}$$

Our goal is to find an assignment of Boolean values to the variables $\{v_k\}$ so that it represents the best possible explanation for the observed sensor values. Clearly, such an assignment should satisfy all linear constraints of the form Eq. (6.5). But, of course, this requirement alone is incomplete since it admits a vacuous solution, e.g., $v_k = 1$ for all k. Therefore, we further qualify our solution with the requirement that it has to minimize the total number of failed sensors. This formalization is based on the intuition that sensors behave nominally most of the time and their failure probabilities are typically much smaller than 0.5. Our formalization also matches the ones popularly used in model-based diagnosis [23]. Of course, richer formalizations can be developed with more information on the prior failure probabilities of individual sensors and physical models of how they interact with each other. Importantly, any preferred formalization can be seamlessly incorporated in our framework.

Overall, we now have the following combinatorial optimization problem for sensor failure detection.

$$\begin{aligned} &\min \sum_{k\in[1,K]} v_k \\ \text{s.t.} &\sum_{k\in[\gamma_n,y_n]} v_k \geq 1, \forall n \in \mathcal{V} \\ &v_k \in \{0,1\}, \forall k \in \{1,2,\cdots,K\} \end{aligned} \tag{6.6}$$

where $\mathcal{V}$ denotes the set of indices of violated constraints. This problem is a specific kind of a 0-1 Integer Linear Program (ILP), called the *Hitting Set Problem*, and is NP-hard to solve in general. However, there are a number of heuristic and approximation algorithms to solve it efficiently. In our implementation, we use the cutting plane method [72] to convert it into a series of Linear Program (LP) relaxations. The basic idea of the cutting plane method is to cut off parts of the feasible region of the LP relaxation, so that the optimal integer solution becomes an extreme point and therefore can be found by the simplex method [92]. It starts by solving the following LP relaxation of (6.6):

$$\begin{aligned} &\min \sum_{k\in[1,K]} v_k \\ \text{s.t.} &\sum_{k\in[\gamma_n,y_n]} v_k \geq 1, \forall n \in \mathcal{V} \\ &0 \leq v_k \leq 1, \forall k \in \{1,2,\cdots,K\} \end{aligned} \tag{6.7}$$

Denote the optimal solution to (6.7) as $\mathbf{v}^*$. For each element in $\mathbf{v}^*$, if its value is already an integer (0 or 1), then we fix its value in the subsequent LP relaxations;

otherwise we treat it as a variable. Now we can solve a new LP relaxation which only contains the variables with fractional values in the solution of the previous LP relaxation. We iterate until all elements of $\mathbf{v}^*$ are integers.

6.2.2 Adapting to Sensor Failures

When sensor failures are detected, we would like the system to automatically adapt to such failures. Our adaptation strategy is to reconstruct the sensor values of the failed sensors from the sensor values of other working sensors. This essentially replaces failed physical sensors with working virtual sensors that enable the system to continue its operation. For reconstructing a failed sensor's values, our approach identifies a constraint in which the output sensor is the failed sensor and all input sensors are working sensors. Then, the corresponding reconstruction function is used. When multiple constraints qualify to be chosen for reconstruction, our procedure selects the constraint with the lowest reconstruction error bound for more accurate results. Specifically, to reconstruct the values of a failed sensor k, we do the following:

1. Find all constraints with working input sensors and output sensor k.
2. Select the constraint with the lowest reconstruction error bound from this set of constraints.
3. Apply the corresponding reconstruction function on the working input sensor values.

6.2.3 Learning Reconstruction Functions from Historical Data

The detection and adaptation procedures discussed above assume that the reconstruction functions are already given. In practice, however, such functions may not be directly available. Instead, we automatically extract them from historical sensor data. Ideally, the learned relationships are expected to have the following properties.

- Accuracy: Each relationship should give us the capability to reconstruct the output sensor value with reasonably low reconstruction error.
- Comprehensiveness: The relationships should be rich enough to help us detect and adapt to various kinds of sensor failures. That is, we would like to extract various types of useful relationships. For example, temperature can be reconstructed using dew point and humidity from the same weather station, and it may also be reconstructed using temperature from nearby weather stations.
- Compactness: The relationships should be easy to state and understand. There are two levels of compactness. First, each relationship should involve only a small number of sensors. The lower the number of sensors, the smaller the chance the constraint is violated. Using a small number of sensors in each relationship improves the overall robustness of our framework and makes the learned relationships more interpretable by humans. Second, the number of learned relationships should also be small since this affects the complexity of our algorithms.

To learn sensor relationships with the above properties, we developed a method that groups input sensors into a number of subsets and then learns reconstruction functions within each subset. The subsets have the following properties.

- Sparsity: Each subset is of small cardinality.
- Disjointness: Subsets tend to be disjoint from each other.

The above properties significantly reduce the number of constraints without compromising the span of what relationships can be represented. In effect, such a subset selection method ensures the compactness and comprehensiveness properties.

Our grouping procedure works as follows. Suppose we want to learn relationships from the input sensor values at the same timestamp $\mathbf{x}_{t,1}, \mathbf{x}_{t,2}, \cdots, \mathbf{x}_{t,K}$ to the output sensor value y_t.[1] To discover the group of input sensors to include in the first constraint, we learn a sparse vector $\mathbf{w}_1 \in \mathcal{R}^K$ that selects a small subset of the input sensors. Mathematically, we solve the following LASSO problem [93].

$$\min_{\mathbf{w}_1} \sum_t (\mathbf{w}_1^{\mathrm{T}} \mathbf{x}_t - y_t)^2 + \lambda \sum_k |\mathbf{w}_{1k}| \tag{6.8}$$

where $\sum_k |\mathbf{w}_{1k}|$ enforces the sparsity of $\mathbf{w}_1$ and $\sum_t (\mathbf{w}_1^{\mathrm{T}} \mathbf{x}_t - y_t)^2$ minimizes the reconstruction error between the linear combination of selected $\mathbf{x}_t$ and y_t over historical data. $\lambda > 0$ is a tradeoff parameter that can be tuned using cross validation.

Once we learn $\mathbf{w}_1$ and identify the relevant subset of the input sensors, a reconstruction function can be learned in many possible ways. To ensure a high quality,

[1]Learning sensor relationships across timestamps is a straightforward generalization.

we apply state-of-the-art nonlinear regression methods (e.g., neural networks) to learn the reconstruction functions from historical data.

$$\min_f \sum_t (y_t - f(\text{subset of } \mathbf{x}_t))^2 \tag{6.9}$$

The subset of input sensors needed for the second constraint can be learned using a new sparse vector $\mathbf{w}_2 \in \mathcal{R}^K$ that is conditioned on $\mathbf{w}_1$. Specifically, we have

$$\min_{\mathbf{w}_2} \sum_t (\mathbf{w}_2^{\mathrm{T}} \mathbf{x}_t - y_t)^2 + \lambda \sum_k |\mathbf{w}_{1k}||\mathbf{w}_{2k}| \tag{6.10}$$

where $\sum_k |\mathbf{w}_{1k}||\mathbf{w}_{2k}|$ encourages $\mathbf{w}_2$ and $\mathbf{w}_1$ to retain a disjoint set of input sensors. As before, we can derive the second reconstruction function from $\mathbf{w}_2$ following Eq. (6.9).

More generally, we can learn the pth vector $\mathbf{w}_p$ that identifies the subset of input sensors in the pth constraint by solving

$$\min_{\mathbf{w}_p} \sum_t (\mathbf{w}_p^{\mathrm{T}} \mathbf{x}_t - y_t)^2 + \lambda \sum_k u_k |\mathbf{w}_{pk}| \tag{6.11}$$

where $u_k = (\sum_{i=1}^{p-1} |\mathbf{w}_{ik}|)/(p-1)$, and $\sum_k u_k |\mathbf{w}_{pk}|$ encourages $\mathbf{w}_p$ to pick sensors different from the ones chosen in the previous $p-1$ subsets. The procedure stops when the reconstruction error in Eq. (6.9) exceeds a pre-defined threshold or p exceeds an upper bound.

Acceleration of Sensor Grouping When the number of sensors K is large, solving a series of Eq. (6.11) instances can be computationally very expensive. As an acceleration strategy, we can first cluster the input sensors into several high-level clusters based on their correlation matrix [100]. After that, we can apply the above grouping procedure within each high-level cluster. Although this strategy ignores possible relationships across high-level clusters, it is computationally attractive and performs well empirically.

6.3 Identifying Modes of Sensor Failures

We consider five common modes of sensor failure [76]:

- Outlier: One or more sensor values are far away from the normal values.
- Spike: A band of consecutive sensor values exhibits a greater-than-expected rate of change.
- Stuck-at: There is zero variation in the sensor values for an unexpected length of time.
- High-noise: There is an unexpectedly high variation in the sensor values in a period of time.
- Miscalibration: There is a constant offset from the ground truth for the sensor values in a period of time.

Identifying the modes of sensor failures is essentially a multi-class classification problem where the input is a time window of sensor values and the output is the identified mode of failure. For such a classification problem, it is important to consider a time window of sensor values because most modes of failure are defined and identifiable only through characteristics of sensor values over a period of time. While existing studies have already explored machine learning techniques like neural networks to classify different modes of failure [27, 80], these methods only capture information from the failed sensor itself. On the other hand, in our approach, we are able to naturally leverage information from multiple related sensors and improve the accuracy and robustness of classification. Specifically, our approach extracts features from the observed sensor values as well as the reconstructed sensor values for a failed sensor. Therefore, the extracted features capture essential information from the failed sensor and the other related working sensors.

Let W be the user-specified size of the time window; and let the observed sensor values of a failed sensor k within such a time window be $[\mathbf{x}_{t-W+1,k}, \mathbf{x}_{t-W+2,k}, \cdots, \mathbf{x}_{t,k}]$. Additionally, let the reconstructed sensor values computed for this failed sensor be $[\hat{\mathbf{x}}_{t-W+1,k}, \hat{\mathbf{x}}_{t-W+2,k}, \cdots, \hat{\mathbf{x}}_{t,k}]$. We first compute informative statistics like the mean, minimum, maximum and standard deviation on both the observed sensor values and the reconstructed sensor values. We then concatenate the raw sensor values and these informative statistics to constitute an input feature vector that can be used to train a classifier. We note that the length of our feature vector depends only on the window size W but not on the number of sensors.

6.4 Results on Weather and Appliance Energy Data

We evaluate our **JDA** framework on sensor data from the weather and appliance energy domains. For both datasets, we use the first half of the time series as historical data required for learning the sensor relationships and the second half of the time series as test data for evaluation. For the weather dataset, the number of sensor failures is fairly small and the modes of sensor failures are also not uniformly distributed. The appliance energy dataset does not contain any sensor failures at all. In order to better evaluate the performances of various algorithms, therefore, we simulate sensor failures in both domains based on a prior history of failures for each sensor. To simulate sensor failures, we run the following procedure multiple times for each sensor.

1. Select any point in the time series with probability 0.01.

2. Starting from each selected point, generate a time window with length chosen uniformly at random from the interval $[1, 30]$. The time window should not overlap with already generated time windows.

3. Select one of the 5 modes of failure uniformly at random.

4. Simulate sensor failures based on the selected mode. Specifically, we generate an instance of each mode of failure in the following ways.

    - Outlier: Set the middle point in the time window to an arbitrary value that significantly deviates from the mean (by more than 3 standard deviations).

    - Spike: Set the middle point in the time window as an outlier; and set the remaining points in the time window using linear interpolation between the middle point and the boundary points.

    - Stuck-at: Set all points in the time window to a fixed arbitrary value.

    - High-noise: Add significant Gaussian noise to all points in the time window.

    - Miscalibration: Offset all points in the time window with a fixed arbitrary bias.

We note that this procedure allows for simultaneous multiple sensor failures windows generated for different sensors can overlap.

Detection of Failures For sensor failure detection, we compare **JDA** to several baseline algorithms.

- **NN**: Nearest Neighbor method, which identifies a sensor failure if the sensor values are far away from normal [67]. This method can detect sensor failure at a vector level (i.e., a group of sensors), but cannot identify which individual sensor(s) actually fail. Therefore we use it for each individual sensor and treat a time window of consecutive readings as a vector. The length of the time window is tuned based on historical data.

- **Subspace**: Subspace method, which learns a set of bases from historical data and then identifies sensor failures if the sensor values are difficult to reconstruct via these bases [64]. Since **Subspace** only identifies sensor failures at a vector level, we adopt the same strategy used in **NN**.

- **Bayesian**: Probabilistic method, which captures linear relationships between sensors and is capable of modeling the working status of each sensor [39].

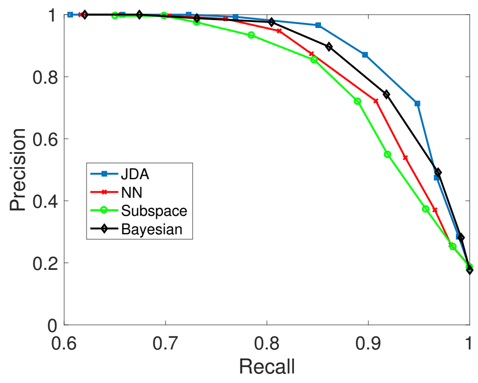
We consider different values of recall (60% to 100%) and measure the corresponding precision. For identifying the modes of failures, we compare **JDA** to the following methods.

- **Neural**: neural networks trained on sensor values from a single sensor.

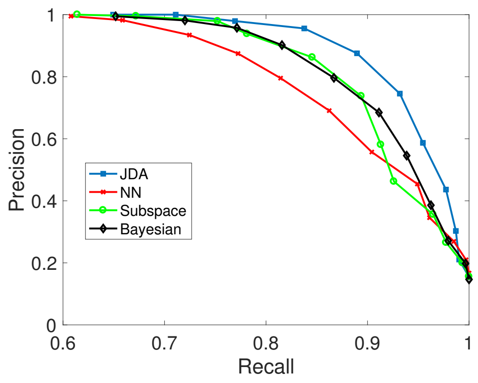
- **Ground**: The same as **JDA**, except that the reconstructed sensor values are replaced by ground truth readings. Although this method is not realistic, it provides an upper bound on the classification accuracy.

In all methods, to classify the mode of failure at time t, we use sensor values in the time window $[t-10, t+10]$. We compute the classification accuracy by comparing the identified mode of failure to the actual mode of failure.

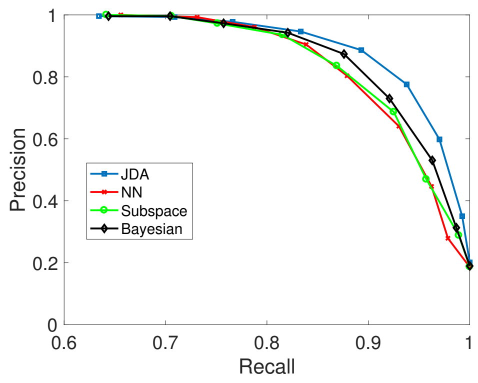
Adaptation to Failures For sensor failure adaptation, we measure both the average reconstruction error and the average excess error of the adaptation error interval. Reconstruction error is measured as the root mean square error (RMSE) between the reconstructed and the ground truth sensor values. For comparison, we introduce a baseline algorithm called **Reference** that uses a simple strategy to reconstruct failed sensor values without using any machine learning techniques. We discuss how **Reference** is implemented for each dataset later. The excess error of the error interval is also measured using the RMSE. We compare the excess error of **JDA** to that of a baseline algorithm called **Const** which uses the constant error bound ϵ_n^2 in Eq. (6.4).



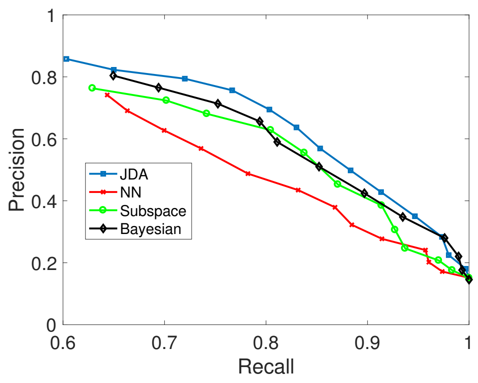
(a) Temperature



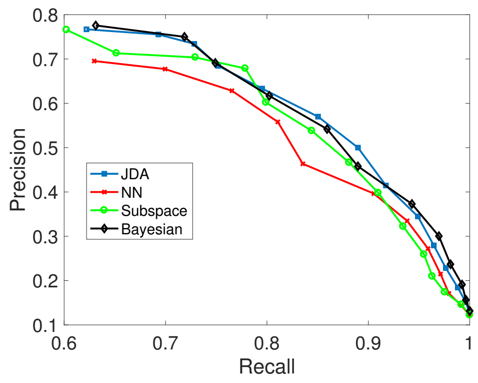
(b) Humidity



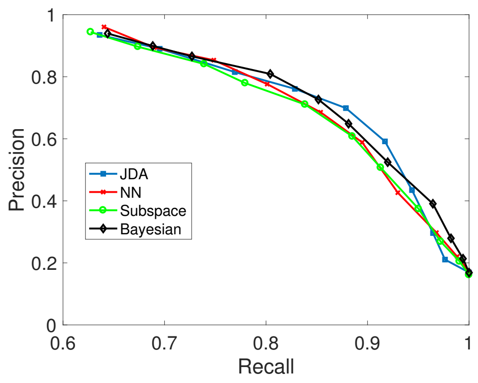
(c) Dew Point



(d) Wind Speed



(e) Wind Gust



(f) Pressure

Figure 6.1: Precision-recall curves on sensor data from the Austin weather stations.

Results on Weather Dataset We use the weather dataset described in Section 2.3. In our experiments, we study 3 nearby stations in San Francisco and 3 nearby

Table 6.1: Accuracy of identifying different modes of sensor failures in the Austin weather stations. Each entry shows the average accuracy and the corresponding standard error. The best performing method between **Neural** and **JDA** (statistically significant up to one standard error) is in bold font.

Sensor	**Neural**	**JDA**	**Ground**
Temperature	92.4 ± 0.5	91.8 ± 0.4	96.4 ± 0.4
Humidity	89.3 ± 0.4	**90.7 ± 0.5**	96.3 ± 0.3
Dew Point	91.6 ± 0.4	91.3 ± 0.5	97.1 ± 0.4
Wind Speed	77.4 ± 1.1	**82.6 ± 0.9**	94.4 ± 0.6
Wind Gust	81.1 ± 1.2	**83.4 ± 1.0**	95.8 ± 0.8
Pressure	85.7 ± 1.0	87.2 ± 0.8	96.0 ± 0.6

Table 6.2: Adaptation performance on sensor data from the Austin weather stations. Each entry shows the average reconstruction error and the corresponding standard error. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	Reconstruction Error		Excess Error	
	Reference	**JDA**	**Const**	**JDA**
Temperature	1.32 ± 0.018	**0.23 ± 0.010**	0.44 ± 0.014	**0.19 ± 0.009**
Humidity	5.28 ± 0.022	**0.41 ± 0.019**	0.93 ± 0.021	**0.30 ± 0.016**
Dew Point	1.15 ± 0.019	**0.33 ± 0.010**	1.21 ± 0.012	**0.17 ± 0.009**
Wind Speed	5.36 ± 0.067	**3.81 ± 0.058**	4.60 ± 0.057	**3.12 ± 0.049**
Wind Gust	5.12 ± 0.061	**3.68 ± 0.050**	5.09 ± 0.048	**2.86 ± 0.042**
Pressure	3.71 ± 0.18	**2.25 ± 0.20**	3.35 ± 0.16	**1.84 ± 0.16**

stations in Austin. For each station, we examine 6 sensors including temperature (°F), humidity (%), dew point (°F), wind speed (mph), wind gust (mph) and pressure (Pa). Sensor values are collected every 5-10 minutes, and data collected over 2 years are used in our experiments.

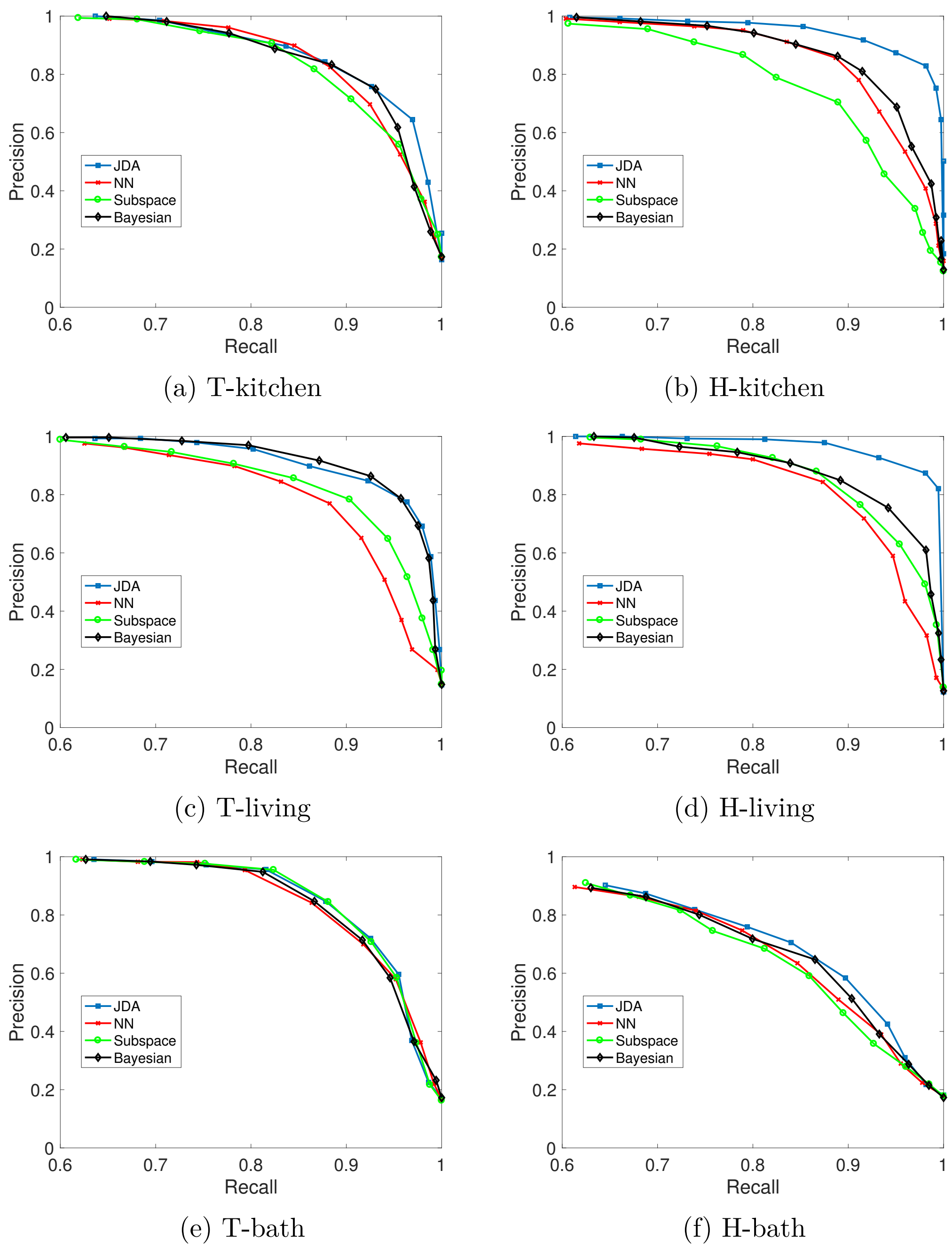
We only show experimental results on the 3 stations in Austin. An explicit discussion of the 3 stations in San Francisco is skipped since these results show very similar trends. We select one station as the target station to evaluate our reconstruction results on. Since there are 3 stations, each sensor type has 3 instances. Due to the spatial proximity of the 3 stations, sensors of the same type are likely to be correlated.

Fig. 6.1 shows the average precision of failure detection on each sensor with recall ranging from 60% to 100%. **JDA** performs the best on all sensors, with a significant margin of improvement on temperature, humidity and dew point. The improvement is less significant on wind speed and wind gust since these signals have relatively large variances and are difficult to reconstruct from other sensor values. When recall is 90%,[2] **JDA** achieves an 8.3% average improvement in precision over all sensors over the second best performer. **Bayesian** performs better than **NN** and **Subspace** on most sensors, showing the benefit of reasoning with multiple sensors. However, **JDA** outperforms **Bayesian** as it captures more nonlinear relationships.

Table 6.1 reports on the accuracy of identifying different modes of sensor failures. Here, **JDA** performs significantly better than **Neural** on three sensors, because **JDA** exploits information from multiple correlated sensors while **Neural** only uses information from a single sensor. **JDA** performs fairly close to **Ground** (except in the case of wind speed and wind gust), highlighting its efficacy in classifying the different modes of sensor failures.

The adaptation performance of **JDA** is given in Table 6.2 where **Reference** is computed as the average RMSE between the sensor values of same sensors in nearby stations. Here, the baseline algorithm is to replace a failed sensor with a similar one from a nearby station. **JDA** achieves significantly lower reconstruction errors than **Reference**, especially on sensors with small variances in their readings. The excess error of **JDA** is consistently better than that of **Const**, which validates our claim that dynamic estimation of error intervals is more accurate than static estimation. It is also easy to see that the excess error of **Const** is relatively large compared to the reconstruction error of **JDA**.

[2]85-95% is a range often used in practice.



(a) T-kitchen (b) H-kitchen

(c) T-living (d) H-living

(e) T-bath (f) H-bath

Figure 6.2: Precision-recall curves on the appliance energy dataset.

Results on the Appliance Energy Dataset The appliance energy dataset consists of 28 sensors measuring energy usage, in-house conditions, and outside

Table 6.3: Accuracy of identifying different modes of sensor failures in the appliance energy domain. Each entry shows the average accuracy and the corresponding standard error. The best performing method between **Neural** and **JDA** (statistically significant up to one standard error) is in bold font.

Sensor	**Neural**	**JDA**	**Ground**
T-kitchen	91.1 $\pm$ 0.5	**92.5 $\pm$ 0.4**	97.7 $\pm$ 0.5
H-kitchen	88.7 $\pm$ 0.7	**93.4 $\pm$ 0.8**	96.4 $\pm$ 0.4
T-living	90.3 $\pm$ 0.8	91.7 $\pm$ 0.7	96.8 $\pm$ 0.7
H-living	87.2 $\pm$ 0.6	**90.6 $\pm$ 0.8**	96.5 $\pm$ 0.7
T-bath	92.6 $\pm$ 0.7	93.8 $\pm$ 0.8	98.0 $\pm$ 0.6
H-bath	82.4 $\pm$ 0.9	**86.2 $\pm$ 0.9**	94.3 $\pm$ 0.7

Table 6.4: Adaptation performance on sensor data from the appliance energy dataset. Each entry shows the average reconstruction error and the corresponding standard error. The best performing method(s) (statistically significant up to one standard error) are in bold font.

Sensor	Reconstruction Error		Excess Error	
	Reference	**JDA**	**Const**	**JDA**
T-kitchen	1.36 $\pm$ 0.024	**0.72 $\pm$ 0.021**	0.75 $\pm$ 0.023	**0.48 $\pm$ 0.020**
H-kitchen	2.85 $\pm$ 0.031	**1.01 $\pm$ 0.019**	1.32 $\pm$ 0.022	**0.83 $\pm$ 0.019**
T-living	1.69 $\pm$ 0.027	**0.80 $\pm$ 0.018**	0.95 $\pm$ 0.023	**0.66 $\pm$ 0.019**
H-living	3.04 $\pm$ 0.036	**1.12 $\pm$ 0.022**	1.34 $\pm$ 0.021	**0.91 $\pm$ 0.016**
T-bath	**0.69 $\pm$ 0.012**	0.73 $\pm$ 0.014	0.85 $\pm$ 0.010	**0.54 $\pm$ 0.011**
H-bath	10.95 $\pm$ 0.068	**8.19 $\pm$ 0.056**	7.93 $\pm$ 0.058	**6.32 $\pm$ 0.049**

conditions.[3] Sensor values are sampled every 10 minutes for about 4.5 months. There are multiple temperature and humidity sensors in different rooms. Their physical proximity leads to strong sensor correlations. In our experiments, we used data from all sensors and report reconstruction results on 6 in-house sensors which measure temperature (°C) and humidity (%) in the kitchen, living room, and bathroom, respectively.

Figure 6.2 shows the average precision of sensor failure detection by different methods, with recall ranging from 60% to 100%. We observe that **JDA** and **Bayesian** perform better than **NN** and **Subspace** in most cases, demonstrating

[3] `http://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction`

that sensor relationships are helpful in detecting sensor failures. On the humidity sensors, H-kitchen and H-living, **JDA** achieves significant improvement even over **Bayesian**, demonstrating the benefit of reasoning with a substrate of constraints and nonlinear relationships proposed in our framework. When recall is 90%, **JDA** achieves a 5.2% average improvement in precision for all sensors over the second best performer.

Table 6.3 reports the accuracy of identifying different modes of sensor failures by different methods, where **JDA** achieves higher accuracy than **Neural** on four sensors. This is because **JDA** exploits information from multiple sensors while **Neural** only uses information from a single sensor.

Table 6.4 reports the adaptation performance, where the recall is set to 90%. To compute **Reference** for a target sensor, we first find the most similar sensor in terms of sensor values from historical data and then calculate the RMSE between the two sensors in the evaluation data. This can be seen as a simple baseline algorithm for adaptation to sensor failures. **JDA** achieves lower reconstruction errors than **Reference** on 5 sensors. However, on the T-bath sensor, **JDA** performs slightly worse than **Reference** due to overfitting. In terms of excess error, **JDA** consistently outperforms **Const**.

6.5 Evaluation in BRASS Project: UUV Results

In BRASS Project [62] Phase 2, we consider the following scenario: a UUV is engaged in a resupply mission to travel to a rendezvous point with a resupply vessel. The evaluation aims to determine our system's capability to adapt to a range of perturbations that affect its ability to localize and determine its position, which is necessary to conduct the resupply mission. These perturbations include failures of sensors used by the UUV as well as perturbations to the UUV environment. Given one or more perturbations, our system needs to adapt the appropriate component. This may involve synthesizing a new data adapter for the sensor in real time and/or reconfiguring higher-layer software components (developed by our collaborator Charles River Analytics).

Test design for this challenge problem involves four failure scenarios:

1. Stuck-at: Consecutive sensor readings that have zero variance and are different from normal sensor readings.

2. High-noise: Consecutive sensor readings that have an unexpectedly high variance.

3. Miscalibration: A time window of sensor readings with a constant offset from the ground-truth values.

4. Chaos: combination of all of the above 3 failures into one execution.

Note that every failure applies to the surge sensor and lasts until the end of the test. Fig. 6.3 visualizes the stuck-at, high-noise, and miscalibration scenarios.

Since the parameter space is very large, we execute 10 tests for each of the failure scenarios that we feel show a meaningful set of results.

To measure the success rate of our system, we define the verdict expression based on how close the UUV comes to the destination. Specifically, if the UUV is less than 75 meters from the destination, the verdict is Pass, else Fail. Thus, a useful way to visualize the verdict is to graphically plot, for each complete test case, the distance from the destination on the x-axis against the test group identifier along the y-axis. We split the graphs into sub-graphs based on each failure scenario (as described above). After running 40 tests (10 for each failure scenario), we plot

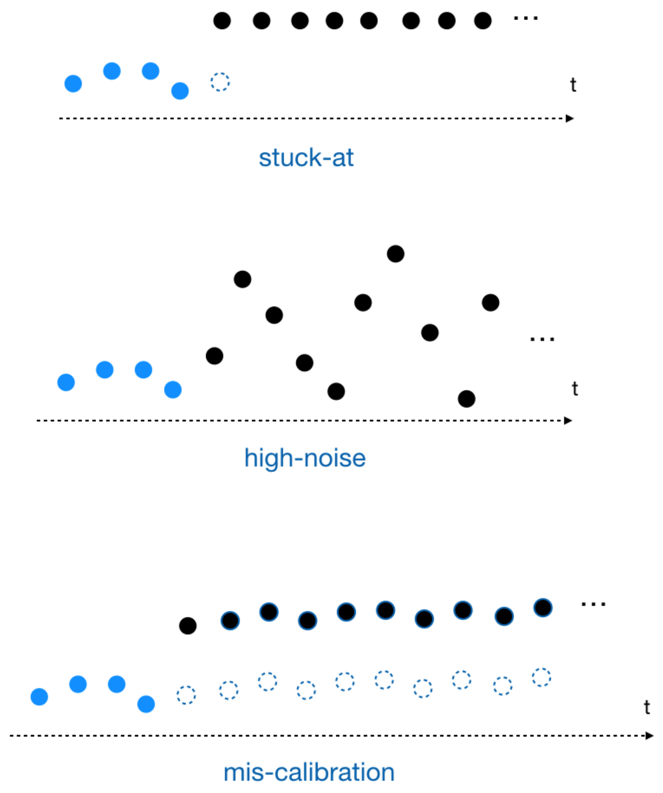


Figure 6.3: Visualization of the stuck-at, high-noise, and miscalibration scenarios.

the results in Fig. 6.4. “Baseline” corresponds to the results without introducing any sensor failure, “Perturbed” corresponds to the results without any adaptation, and “Adapted” corresponds to the results with adaptation. From Fig. 6.4, we can see that for the stuck-at, high-noise and chaos scenarios, Adapted performs better than Perturbed and is within the required 75 meters (vertical blue line) in most tests. For the high-noise scenario, Adapted performs very similarly to Perturbed, and both are within the required 75 meters. Overall, the verdict is Pass in 90% of the tests. This demonstrates that our system can adapt to sensor failures with high efficacy and robustness.

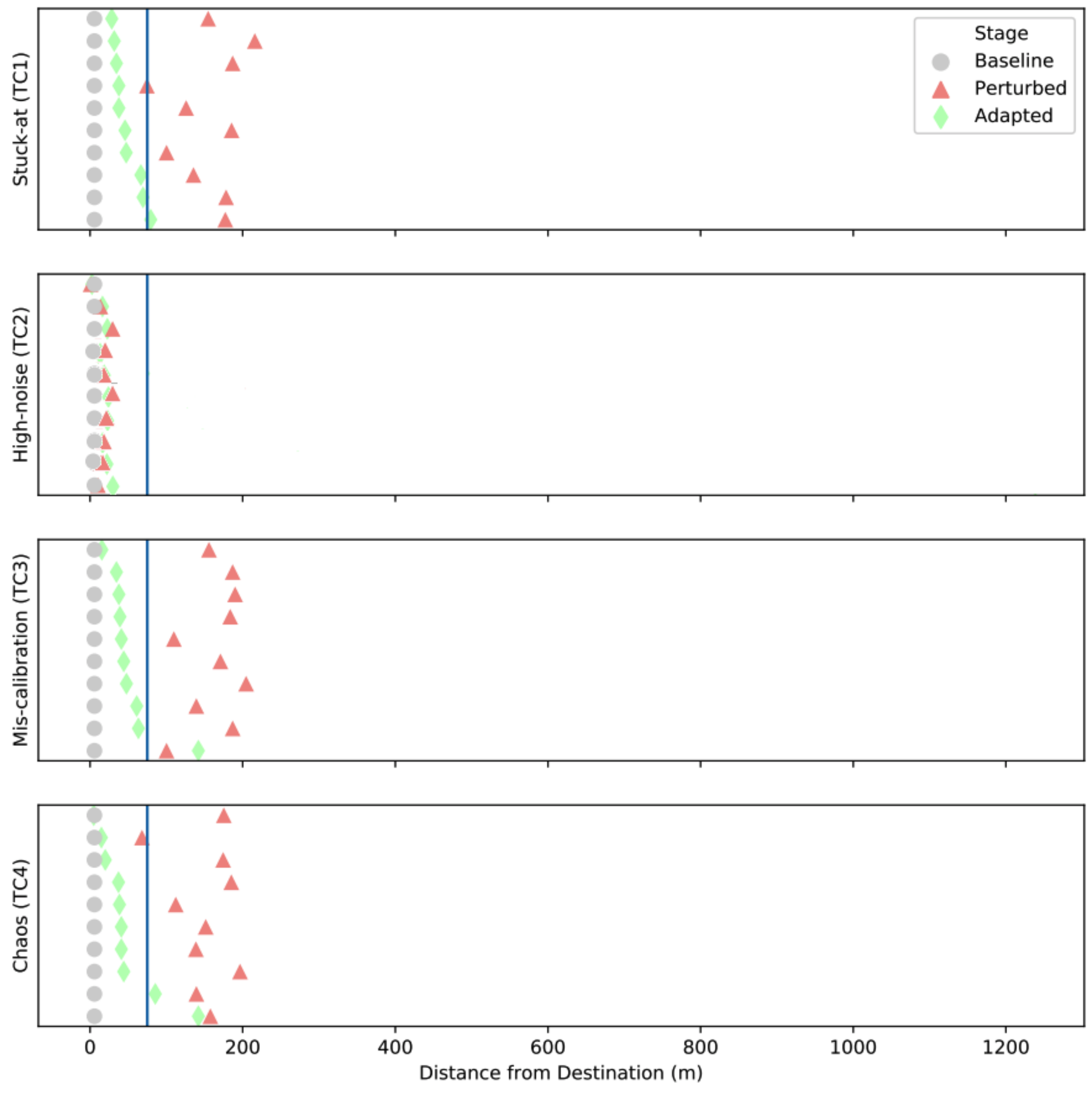


Figure 6.4: UUV distances for all tests separated by each failure scenario. The vertical blue line indicates 75 meters, as required for a Pass verdict.

Chapter 7

Related Work

7.1 Detecting Sensor Failures and Changes

Sensor failures and changes can be detected by identifying abrupt changes in time series of sensor readings. This problem is often called change point detection which attracted researchers in statistics and data mining communities for decades [10, 53, 17, 3, 79]. Change point detection has broad applications in fraud detection, network intrusion detection, motion detection in vision, fault detection in controlled systems, etc.

Change point detection methods can be categorized into two types: supervised and unsupervised. Supervised methods treat change point detection as a classification problem and classify sensor readings into different states learned from training data. Researchers have developed a number of supervised methods based on support vector machines [83, 98], nearest neighbors [83, 104], Gaussian mixture models [28, 54], etc.

The limitation of supervised methods is that they require training data for all possible states or classes. Unsupervised methods, on the other hand, are capable of handling a variety of different states without prior training for each state. Existing unsupervised methods can be further classified into the following categories

- Distribution-based methods, which identify a change point if data distributions before and after that point are significantly different [65, 101, 55].

- Reconstruction-based methods, which attempt to reconstruct a data point using neural networks [29, 57, 90] or bases learned by subspace methods [64, 74, 63, 20]. If a data point is hard to reconstruct well, then it is detected as a change point.

- Probabilistic methods, which compute the likelihood of a data point through Bayesian networks and identify a change point if the likelihood is below a threshold [1, 84, 37, 39, 36].

- Distance-based methods, which identify a change point by examining its distance to other points. Existing work explore nearest neighbor distances [5, 12, 21], clustering structures [66, 9, 18] and graph structures [22] derived from distances.

Some of the existing approaches work for a single sensor, without leveraging other related sensors. For approaches that take advantage of multiple sensors, reconstruction-based ones are most similar to our approach, conceptually. Our approach attempts to reconstruct sensor readings by exploiting different types of relationships among sensors, and detecting sensor failures or changes if the observed readings are different from reconstructed ones. Exploiting different types of sensor relationships makes our approach more accurate and robust. It can identify exact failed or changed individual sensors from a group of sensors while existing reconstruction-based approaches often detect changes at a group level. Additionally, our approach can leverage available sensors in a dynamic manner, without assuming that all sensors are always working in the training set.

Probabilistic approaches [58, 37, 39, 36] are capable of identifying changes in individual sensors because they model states of each sensor using dynamic Bayesian networks [35]. However, existing work only model linear relationships among sensors. In contrast, our work enables exploiting nonlinear relationships that often exist in the real world.

7.2 Reconstruction of Sensor Readings

Existing work examining sensor failures and changes mainly focus on detecting change points but rarely address the issue of adaptation to sensor failures or changes. Typically, they rely on human experts to examine these change points and make subsequent decisions. Our work, on the other hand, is motivated by the notion of survivable software and aims at automatic adaptation to changes. Although some of the existing detection methods [37, 39, 36] can be used to reconstruct sensor readings because they infer the actual readings through their models, they are not able to leverage any new sensor.

Our work adapts to sensor failures and changes by learning functions to reconstruct the original sensor readings. When dealing with sensor failures, learning reconstruction functions is equivalent to regression problems where we use a method called Fast Function Extraction (FFX) [73]. Compared to other methods to learn functions, such as linear regression [43], kernel ridge regression [75] and neural networks [4], FFX is capable of learning compact nonlinear function forms efficiently, and provides more human interpretation of the learned forms.

7.3 Domain Adaptation

Our model-level adaptation can be viewed in the framework of domain adaptation [33, 77, 82] which addresses learning problems with mismatched distributions. The *source* domain refers to the labeled training data, while the *target* domain refers to the test data. When there are no labeled data from the target domain to help learning classifiers[1], the problem setting is called *unsupervised domain adaptation*. Over the past decade, a number of unsupervised domain adaptation approaches have been developed and used in applications such as computer vision [30], natural language processing [77], sensor data analysis [78], etc. Recently, the work of Purushotham et al. [81] studies the adaptation problem in classifying time series.

Unsupervised domain adaptation is especially challenging as the target domain does not explicitly provide any information on how to optimize classifiers. Note that the objective of domain adaptation is to derive a classifier for the unlabeled (target) data from the labeled (source) data. This goal sets domain adaptation apart from semi-supervised learning, whose primary goal is to improve the performance on the labeled data with unlabeled data [19]. The difference is subtle yet fundamental. For example, model selection or cross-validation using classification accuracy on the target domain is generally impossible.

Most existing approaches [78, 49, 48, 24] for unsupervised domain adaptation follow a two-stage learning paradigm: they first identify a domain-invariant feature space such that the marginal distributions of the two domains are the same, and then learn a classifier in that space. For example, in covariate shift [89, 14, 61], the labeled instances from the source domain are first weighted so as to compensate for the difference in marginal distributions. Then, a classifier is trained using the labels and later applied to the unlabeled data. In structural correspondence learning, the original features are first augmented with features that are more likely to be domain-invariant; then a classifier is trained [16]. The augmenting features

[1]Most domain adaptation approaches involve learning classifiers, but the methodologies can also be applied to regression models.

are a linear transformation of the original features. Alternatively, in deep learning architecture for domain adaptation, the augmenting features are a highly nonlinear transformation of the original ones [45, 24].

Underlying these methods is the assumption that there exists a domain-invariant feature space and that classifiers learned in the new space will perform equally well on both domains. However, maximizing the similarity in marginal distributions may not bear a direct consequence on (dis)similarities between posterior distributions. As an extreme case, projecting features into irrelevant feature dimensions would make the two domains look very much alike. This motivates a single-stage learning paradigm that jointly learns the domain-invariant feature space and the classifiers. For instance, the work in [31, 7, 8, 95, 44] optimize classification performance on the source domain while learning the domain-invariant feature space. Different from existing single-stage approaches that optimize source-domain classifiers, our work directly optimizes target-domain classifiers [88]. We consider this as an important hallmark of our approach because optimizing classifiers on the target domain is our primary objective, and purely optimizing classifiers on the source domain may lead to overfitting.

Heterogeneous domain adaptation. When the feature spaces of the source and target domains are different, the problem setting is called heterogeneous domain adaptation. In the scenario of sensor changes, if one original sensor is replaced with multiple new sensors, then model-level adaptation becomes an instance of heterogeneous domain adaptation. Previous work on heterogeneous domain adaptation mainly consists of two types of approaches. One type learns a transformation to map the features from one domain to the other [32, 91, 105] by leveraging sample-level correspondences across domains (e.g., an image and its tag). A second type of approach maps the source and target domains into a domain-invariant feature space in which their marginal distributions are similar, and then learns a model in that space using labeled data [68, 96, 6, 40, 86, 56, 97, 102, 26]. As part of the thesis, we extend our work [88] to heterogeneous feature spaces following the second type of approach.

Our sensor-level adaptation can also be viewed as an instance of heterogeneous domain adaptation if we treat the replaced sensor readings as labels and the readings of other sensors as features [87]. However, existing heterogeneous domain adaptation approaches are not capable of solving our problem where the target domain has new features that are unseen in the source domain, as discussed in Chapter 3.

Chapter 8

Conclusion

In this thesis, we studied how to automatically adapt to failures and changes in sensors, which is an important problem in building survivable software. We proposed a series of adaptation approaches for addressing failures and changes in both individual and compound sensors. Our approaches have the following novel capabilities.

- They enable two levels of adaptation: sensor-level and model-level.
- They enable adaptation to new sensors when there is no overlapping period between the new sensors and the replaced sensors.
- They leverage sensor-specific transformations derived from historical sensor data.
- They leverage spatial and temporal information about sensors to improve the robustness and accuracy of adaptations.
- They can scale to a large number of reference sensors and new sensors.
- They estimate the quality of adaptation that is useful for higher-layer software.
- They use a constraint-based framework for joint detection and adaptation to sensor failures.

To validate our approaches, we conducted experiments on sensor data from the weather and UUV domains. Our empirical results demonstrate that our approaches can automatically detect and adapt to sensor failures and changes with higher accuracy and robustness compared to other alternative approaches.

Our work is of highest relevance to researchers and practitioners working in the areas of Software Systems, Internet of Things and Machine Learning.

Discussion. Note that for our sensor-level adaptation approaches, the underlying assumption is that sensor values from a subset of sensors are well correlated. Although this assumption often holds in real-world systems, it may not always be the case. This can be viewed as a limitation of sensor-level adaptation when the correlations among sensors are weak. However, such a limitation can often be overcome in practice if we are allowed to access or install more reference sensors that are better correlated with existing sensors. Also, when sensor-level adaptation is challenging, model-level adaptation may still work if our goal is to directly adapt software components built on the sensor values.

Future work. We would like to explore two directions in our future work. First is to apply our approaches to new domains with larger volumes of sensor data. For example, we plan to examine the helicopter domain where sensor values are sampled in milliseconds. Second is to integrate our approaches into survivable software systems that operate in real-world scenarios. We are in the process of deploying our approaches into a real UUV and testing it in ocean waters.

Reference List

[1] Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[2] Oleg A Alduchov and Robert E Eskridge. Improved magnus form approximation of saturation vapor pressure. *Journal of Applied Meteorology*, 35(4):601–609, 1996.

[3] Samaneh Aminikhanghahi and Diane J Cook. A survey of methods for time series change point detection. *Knowledge and Information Systems*, pages 1–29, 2016.

[4] Nikolaos Ampazis and Stavros J Perantonis. Two highly efficient second-order algorithms for training feedforward networks. *IEEE Transactions on Neural Networks*, 13(5):1064–1074, 2002.

[5] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 15–27. Springer, 2002.

[6] Andreas Argyriou, Andreas Maurer, and Massimiliano Pontil. An algorithm for transfer learning in a heterogeneous environment. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 71–85. Springer, 2008.

[7] Mahsa Baktashmotlagh, Mehrtash T Harandi, Brian C Lovell, and Mathieu Salzmann. Unsupervised domain adaptation by domain invariant projection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 769–776, 2013.

[8] Mahsa Baktashmotlagh, Mehrtash T Harandi, Brian C Lovell, and Mathieu Salzmann. Domain adaptation on the statistical manifold. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2481–2488, 2014.

[9] Daniel Barbará, Yi Li, and Julia Couto. Coolcat: an entropy-based algorithm for categorical clustering. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 582–589. ACM, 2002.

[10] Michèle Basseville, Igor V Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs, 1993.

[11] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. *ECCV*, 2006.

[12] Stephen D Bay and Mark Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 29–38. ACM, 2003.

[13] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. Analysis of representations for domain adaptation. *NIPS*, 2007.

[14] S. Bickel, M. Brückner, and T. Scheffer. Discriminative learning for differing training and test distributions. In *Prof. of ICML*, pages 81–88, 2007.

[15] J. Blitzer, M. Dredze, and F. Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proc. of ACL*, 2007.

[16] J. Blitzer, R. McDonald, and F. Pereira. Domain adaptation with structural correspondence learning. In *Proc. of EMNLP*, pages 120–128. Association for Computational Linguistics, 2006.

[17] E Brodsky and Boris S Darkhovsky. *Nonparametric methods in change point problems*, volume 243. Springer Science & Business Media, 2013.

[18] Suratna Budalakoti, Ashok N Srivastava, Ram Akella, and Eugene Turkov. Anomaly detection in large sets of high-dimensional symbol sequences. 2006.

[19] O. Chapelle, B. Schölkopf, A. Zien, et al. *Semi-supervised learning*, volume 2. MIT press Cambridge, MA:, 2006.

[20] Vasilis Chatzigiannakis, Symeon Papavassiliou, Mary Grammatikou, and B Maglaris. Hierarchical anomaly detection in distributed large-scale sensor networks. In *Computers and Communications, 2006. ISCC'06. Proceedings. 11th IEEE Symposium on*, pages 761–767. IEEE, 2006.

[21] Sanjay Chawla and Pei Sun. Slom: a new measure for local spatial outliers. *Knowledge and Information Systems*, 9(4):412–429, 2006.

[22] Hao Chen, Nancy Zhang, et al. Graph-based change-point detection. *The Annals of Statistics*, 43(1):139–176, 2015.

[23] Jie Chen and Ron J Patton. *Robust model-based fault diagnosis for dynamic systems*, volume 3. Springer Science & Business Media, 2012.

[24] Minmin Chen, Zhixiang Xu, Kilian Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. *arXiv preprint arXiv:1206.4683*, 2012.

[25] Sida Chen, Shigeru Imai, Wennan Zhu, and Carlos A Varela. Towards learning spatio-temporal data stream relationships for failure detection in avionics. *Dynamic Data-Driven Application Systems (DDDAS 2016), Hartford, CT*, 2016.

[26] Wei-Yu Chen, Tzu-Ming Harry Hsu, Yao-Hung Hubert Tsai, Yu-Chiang Frank Wang, and Ming-Syan Chen. Transfer neural trees for heterogeneous domain adaptation. In *European Conference on Computer Vision*, pages 399–414. Springer, 2016.

[27] Anders Lyhne Christensen, Rehan OâĂŹGrady, Mauro Birattari, and Marco Dorigo. Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, 24(1):49–67, 2008.

[28] Ian Cleland, Manhyung Han, Chris Nugent, Hosung Lee, Sally McClean, Shuai Zhang, and Sungyoung Lee. Evaluation of prompted annotation of activity data recorded from a smart phone. *Sensors*, 14(9):15861–15879, 2014.

[29] Paul A Crook, Stephen Marsland, Gillian Hayes, and Ulrich Nehmzow. A tale of two filters-on-line novelty detection. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, pages 3894–3899. IEEE, 2002.

[30] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. *arXiv preprint arXiv:1702.05374*, 2017.

[31] Gabriela Csurka, Boris Chidlowskii, Stéphane Clinchant, and Sophia Michel. Unsupervised domain adaptation with regularized domain instance denoising. In *Computer Vision–ECCV 2016 Workshops*, pages 458–466. Springer, 2016.

[32] Wenyuan Dai, Yuqiang Chen, Gui-Rong Xue, Qiang Yang, and Yong Yu. Translated learning: Transfer learning across different feature spaces. In *Advances in neural information processing systems*, pages 353–360, 2008.

[33] H. Daumé III and D. Marcu. Domain adaptation for statistical classifiers. *JAIR*, 26:101–126, 2006.

[34] Hal Daume III and Daniel Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 26:101–126, 2006.

[35] Thomas L Dean and Keiji Kanazawa. Probabilistic temporal reasoning. In *AAAI*, pages 524–529, 1988.

[36] Ethan W Dereszynski and Thomas G Dietterich. Spatiotemporal models for data-anomaly detection in dynamic environmental monitoring campaigns. *ACM Transactions on Sensor Networks (TOSN)*, 8(1):3, 2011.

[37] Ethan W Dereszynski and Thomas G Dietterich. Probabilistic models for anomaly detection in remote sensor data streams. *arXiv preprint arXiv:1206.5250*, 2012.

[38] I.S. Dhillon, S. Mallela, and D.S. Modha. Information-theoretic co-clustering. In *Proc. of SIGKDD*, 2003.

[39] Thomas G Dietterich, Ethan W Dereszynski, Rebecca A Hutchinson, and Daniel R Sheldon. Machine learning for computational sustainability. In *IGCC*, page 1, 2012.

[40] Lixin Duan, Dong Xu, and Ivor W Tsang. Learning with augmented features for heterogeneous domain adaptation. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 711–718, 2012.

[41] Eiman Elnahrawy and Badri Nath. Context-aware sensors. In *European Workshop on Wireless Sensor Networks*, pages 77–93. Springer, 2004.

[42] J.W. Fisher III and J.C. Principe. A methodology for information theoretic feature extraction. In *Proc. IEEE World Congress on Comp. Intell.*, 1998.

[43] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[44] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016.

[45] X. Glorot, A. Bordes, and Y. Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proc. of ICML*, 2011.

[46] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. *NIPS*, 2004.

[47] R. Gomes, A. Krause, and P. Perona. Discriminative clustering by regularized information maximization. In *NIP*, 2010.

[48] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2066–2073. IEEE, 2012.

[49] R. Gopalan, R. Li, and R. Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *Proc. of ICCV*, 2011.

[50] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. *NIPS*, 17:529–236, 2005.

[51] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical report, California Institute of Technology, 2007.

[52] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[53] Fredrik Gustafsson and Fredrik Gustafsson. *Adaptive filtering and change detection*, volume 1. Citeseer, 2000.

[54] Manhyung Han, Young-Koo Lee, Sungyoung Lee, et al. Comprehensive context recognizer based on multimodal sensors in a smartphone. *Sensors*, 12(9):12588–12605, 2012.

[55] Zaid Harchaoui, Eric Moulines, and Francis R Bach. Kernel change-point analysis. In *Advances in neural information processing systems*, pages 609–616, 2009.

[56] Maayan Harel and Shie Mannor. Learning from multiple outlooks. 2010.

[57] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier detection using replicator neural networks. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 170–180. Springer, 2002.

[58] David J Hill, Barbara S Minsker, and Eyal Amir. Real-time bayesian anomaly detection for environmental sensor data. In *Proceedings of the Congress-International Association for Hydraulic Research*, volume 32, page 503. Citeseer, 2007.

[59] G. Hinton and S.T. Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15:833–840, 2002.

[60] Chenping Hou and Zhi-Hua Zhou. One-pass learning with incremental and decremental features. *arXiv preprint arXiv:1605.09082*, 2016.

[61] J. Huang, A.J. Smola, A. Gretton, K.M. Borgwardt, and B. Scholkopf. Correcting sample selection bias by unlabeled data. *NIPS*, 19:601, 2007.

[62] Jeffrey Hughes, Cassandra Sparks, Alley Stoughton, Rinku Parikh, Albert Reuther, and Suresh Jagannathan. Building resource adaptive software systems (brass): Objectives and system evaluation. *ACM SIGSOFT Software Engineering Notes*, 41(1):1–2, 2016.

[63] Tsuyoshi Idé and Keisuke Inoue. Knowledge discovery from heterogeneous dynamic systems using change-point correlations. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 571–575. SIAM, 2005.

[64] Tsuyoshi Idé and Koji Tsuda. Change-point detection using krylov subspace learning. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 515–520. SIAM, 2007.

[65] Yoshinobu Kawahara and Masashi Sugiyama. Sequential change-point detection based on direct density-ratio estimation. *Statistical Analysis and Data Mining*, 5(2):114–127, 2012.

[66] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 289–296. IEEE, 2001.

[67] Edwin M Knox and Raymond T Ng. Algorithms for mining distancebased outliers in large datasets. In *Proceedings of the International Conference on Very Large Data Bases*, pages 392–403. Citeseer, 1998.

[68] Brian Kulis, Kate Saenko, and Trevor Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1785–1792. IEEE, 2011.

[69] Ted Tsung-Te Lai, Wei-Ju Chen, Kuei-Han Li, Polly Huang, and Hao-Hua Chu. Triopusnet: Automating wireless sensor network deployment and replacement in pipeline monitoring. In *Proceedings of the 11th international conference on Information Processing in Sensor Networks*, pages 61–72. ACM, 2012.

[70] Mark G Lawrence. The relationship between relative humidity and the dew-point temperature in moist air: A simple conversion and applications. *Bulletin of the American Meteorological Society*, 86(2):225–233, 2005.

[71] Y. Mansour, M. Mohri, and A. Rostamizadeh. Domain adaptation: Learning bounds and algorithms. *Proc. of COLT*, 2009.

[72] Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1-3):397–446, 2002.

[73] Trent McConaghy. Ffx: Fast, scalable, deterministic symbolic regression technology. In *Genetic Programming Theory and Practice IX*, pages 235–260. Springer, 2011.

[74] Valentina Moskvina and Anatoly Zhigljavsky. An algorithm based on singular spectrum analysis for change-point detection. *Communications in Statistics-Simulation and Computation*, 32(2):319–352, 2003.

[75] Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.

[76] Kevin Ni, Nithya Ramanathan, Mohamed Nabil Hajj Chehade, Laura Balzano, Sheela Nair, Sadaf Zahedi, Eddie Kohler, Greg Pottie, Mark Hansen, and Mani Srivastava. Sensor network data fault types. *ACM Transactions on Sensor Networks (TOSN)*, 5(3):25, 2009.

[77] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[78] S.J. Pan, I.W. Tsang, J.T. Kwok, and Q. Yang. Domain adaptation via transfer component analysis. *IEEE Trans. Neur. Nets.*, 22(2):199, 2011.

[79] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[80] Piotr Przystałka. Model-based fault detection and isolation using locally recurrent neural networks. In *International Conference on Artificial Intelligence and Soft Computing*, pages 123–134. Springer, 2008.

[81] Sanjay Purushotham, Wilka Carvalho, Tanachat Nilanon, and Yan Liu. Variational recurrent adversarial deep domain adaptation. *International Conference on Learning Representations*, 2017.

[82] J. Quiñonero-Candela, M. Sugiyama, and A. Schwaighofer. *Dataset Shift in Machine Learning.* The MIT Press, 2009.

[83] Sasank Reddy, Min Mun, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Using mobile phones to determine transportation modes. *ACM Transactions on Sensor Networks (TOSN)*, 6(2):13, 2010.

[84] Yunus Saatçi, Ryan D Turner, and Carl E Rasmussen. Gaussian process change point models. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 927–934, 2010.

[85] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *Proc. of ECCV*, 2010.

[86] Xiaoxiao Shi, Qi Liu, Wei Fan, S Yu Philip, and Ruixin Zhu. Transfer learning on heterogenous feature spaces via spectral transformation. In *2010 IEEE international conference on data mining*, pages 1049–1054. IEEE, 2010.

[87] Yuan Shi and Craig Knoblock. Learning with previously unseen features. *IJCAI*, 2017.

[88] Yuan Shi and Fei Sha. Information-theoretical learning of discriminative clusters for unsupervised domain adaptation. *ICML*, 2012.

[89] H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *J. of Stat. Plan. and Infer.*, 90(2):227–244, 2000.

[90] Sameer Singh and Markos Markou. An approach to novelty detection applied to the classification of image regions. *IEEE Transactions on Knowledge and Data Engineering*, 16(4):396–407, 2004.

[91] Richard Socher, Milind Ganjoo, Christopher D Manning, and Andrew Ng. Zero-shot learning through cross-modal transfer. In *Advances in neural information processing systems*, pages 935–943, 2013.

[92] Daniel Solow. Linear and nonlinear programming. *Wiley Encyclopedia of Computer Science and Engineering*, 2007.

[93] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B*, 1996.

[94] Bin Tong, Guiling Wang, Wensheng Zhang, and Chuang Wang. Node reclamation and replacement for long-lived sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 22(9):1550–1563, 2011.

[95] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4068–4076, 2015.

[96] Chang Wang and Sridhar Mahadevan. Heterogeneous domain adaptation using manifold alignment. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1541, 2011.

[97] Bin Wei and Christopher J Pal. Heterogeneous transfer learning with rbms. In *AAAI*, 2011.

[98] Li Wei and Eamonn Keogh. Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 748–753. ACM, 2006.

[99] K.Q. Weinberger and L.K. Saul. Distance metric learning for large margin nearest neighbor classification. *JMLR*, 10:207–244, 2009.

[100] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[101] Kenji Yamanishi and Jun-ichi Takeuchi. A unifying framework for detecting outliers and change points from non-stationary time series data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 676–681. ACM, 2002.

[102] Yi-Ren Yeh, Chun-Hao Huang, and Yu-Chiang Frank Wang. Heterogeneous domain adaptation and classification by exploiting the correlation subspace. *IEEE Transactions on Image Processing*, 23(5):2009–2018, 2014.

[103] Peilin Zhao and Steven C Hoi. Otl: A framework of online transfer learning. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 1231–1238, 2010.

[104] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *Proceedings of the 17th international conference on World Wide Web*, pages 247–256. ACM, 2008.

[105] Joey Tianyi Zhou, Sinno Jialin Pan, Ivor W Tsang, and Yan Yan. Hybrid heterogeneous transfer learning through deep learning. In *AAAI*, pages 2213–2220, 2014.

[106] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.