Improving machine learning methods for solving non-stationary conditions based on data availability, time urgency, and types of change

Goh, Chun Fan

2021

Goh, C. F. (2021). Improving machine learning methods for solving non-stationary conditions based on data availability, time urgency, and types of change. Doctoral thesis, Nanyang Technological University, Singapore. <https://hdl.handle.net/10356/147041>

<https://hdl.handle.net/10356/147041>

<https://doi.org/10.32657/10356/147041>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 20:12:33 SGT

Improving Machine Learning Methods for Solving Non-Stationary Conditions Based on Data Availability, Time Urgency, and Types of Change

Goh Chun Fan

School of Mechanical and Aerospace Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2021

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

29 December 2019

.....

Date

.....

Goh Chun Fan

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

30 December 2019

.....

Date

.....

Prof. Gerald Seet Gim Lee

Authorship Attribution Statement

This thesis contains material from four papers published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 6 is published as [Goh, Chun Fan, Gerald Gim Lee Seet, and Kenji Shimada](#). "A model-based learning controller with predictor augmentation for non-stationary conditions and time delay in water shooting." *Automation Science and Engineering (CASE)*, 2017 13th IEEE Conference on. IEEE, 2017. DOI: [10.1109/COASE.2017.8256253](#).

The contributions of the co-authors are as follows:

- Prof. Shimada and Prof. Seet provided the initial project direction.
- I prepared the manuscript drafts. The manuscript was revised by Prof. Shimada and Prof. Seet.
- I designed the study and performed all the laboratory work at the Computational Engineering and Robotics Laboratory (CERLAB) at Carnegie Mellon University. I also analyzed the data.

Chapter 7 is published as [Goh, Chun Fan, Laura B. Samuelsson, Martica H. Hall, Gerald Gim Lee Seet, and Kenji Shimada](#). "Semi-Automatic Snore Detection in Polysomnography based on Hierarchical Clustering." *Automation Science and Engineering (CASE)*, 2018 14th IEEE Conference on. IEEE, 2018. DOI: [10.1109/COASE.2018.8560510](#).

The contributions of the co-authors are as follows:

- Ms. Samuelsson and Prof. Hall provided the data and the initial project direction.
- I wrote the drafts of the manuscript. The manuscript was revised together by Prof. Shimada and Prof. Seet.
- I designed and coded the software, ran the experiment, and conducted the analysis.

The results in Chapter 8 is first presented in [Akhil Chaturvedi, Chun Fan Goh, Rahul Shailesh Raina, and Kenji Shimada](#). "Comparing Deep Feature Representations to Improve Robustness to Subject Variation in Snore Detection", Poster Session, SLEEP 2019. DOI: [10.1093/sleep/zsz067.448](#).

The contributions of the co-authors are as follows:

- Prof. Hall and I provided the initial project direction.

- Mr. Chaturvedi, Mr. Raina, and I programmed the software and ran the experiments.
- Mr. Chaturvedi and I conducted the results analysis.
- Mr. Chaturvedi and I prepared the poster. The poster was revised by Prof. Shimada.

The mobility design presented in Chapter 9 is published in [Chun Fan Goh, Akshay Hinduja, Divish Ajmani, Robin Song, Lei Zhang, Kenji Shimada](#). “Designing a Mobility Solution for Fully Autonomous Double Hull Block Welding”. ASME Journal of Mechanism and Robotics. DOI: 10.1115/1.4043604.

The contributions of the co-authors are as follows:

- Prof. Shimada and I provided the initial project direction and the conceptual design.
- Ms. Zhang and I designed the robot prototype.
- Ms. Zhang, Ms. Song, Mr. Ajmani, and I constructed the robot prototype.
- Ms. Zhang, Ms. Song, Mr. Ajmani, and I programmed the robot and ran the experiments.
- Mr. Hinduja and I conducted the results analysis.
- Mr. Hinduja and I prepared the manuscript. The manuscript was revised by Prof. Shimada.

29 December 2019

.....

Date

.....

Goh Chun Fan

Acknowledgements

I would like to thank my thesis advisors, Dr. Kenji Shimada and Dr. Gerald Seet for their guidance and mentoring. It has been an honor to work under them.

Many thanks to my laboratory mates, Dr. Ying Ying Wu, Mabaran Rajaraman, Dr. Meghan Chandarana, Dr. Ved Vyas, Sooho Park, Di Deng, Dr. Wei Jing, Ryusuke Kubo, Dr. Soji Yamakawa, and Tomotake Furuhashi for their useful discussion and companionship. They made the endeavor more bearable.

To my family, thank you for your unwavering support, patience, and indulgence. Your love is the reason I persevered.

Thank you to my friends back home in Malaysia. They make me feel like an MVP whenever I go home. In particular, King Hong for his constant intellectual stimuli, Chee Yoong for his gastronomical treats, Kah Choon for being my mentor in relationships, Jen Terng for being my drinking buddy, and Quai Ling for entertaining me with her endless shopping spree.

I am also grateful to my friends in Singapore. Specifically, Fifi for his amusing love stories, Tyng Tze and Chee Lap for letting me stay over whenever I visit, Cheong for having my durian craving fixed, and Kok Seong for having faith in me. I also want to thank the Ashdown Gang: Fitri, Boon, and Yifeng for their timely and life-changing advice when I was lost.

My work in the snore detection project was assisted by Akhil Chaturvedi, Yuzhong Hua, and Rahul Shailesh Raina. I want to thank them for the help.

My work in the welding robot project was assisted by Lei Zhang, Yoichi Kuwano, Nishi Nobuaki, Kushal Sandeep Doshi, Cameron Stabile, Adam Chamberlin, Arjun Suresh, Shishir Gupta, Ishraq Bhuiyan, Kwadwo Akompong, Julia Lu, Shuming Teoh, Sarah Chen, Brendan Wixen, Tomasz Swierzewski, Kyshalee Vazquez-Santiago, Qindong Zhang, Hai Duy Tran, Robin Song, Divish Ajmani, Akshay

Hinduja, Xiao Jiang, Alex Sun, Kashyap Adury, Advait Ajay Deshpande, and Anish Deva. I thank them for their assistance.

This work was partially supported by the Singapore Economic Development Board (EDB), Tsuneishi Shipbuilding Co. Ltd., and Philips Respirationics.

“Unfortunately practice generally precedes theory, and it is the usual fate of mankind to get things done in some boggling way first, and find out afterward how they could have been done much more easily and perfectly.”

—Charles Sanders Peirce

To my parents,
Goh Lai Chon & Lee Koon Foong

Abstract

Supervised learning algorithms do not work well when the deployment condition is dissimilar to the training condition. Such non-stationary conditions include covariate shifts and concept shifts. Importance weighted learning (IWL) is used to handle a one-time covariate shift but not frequent shifts and concept shifts. While forgetting addresses concept shifts, it is wasteful in discarding previously learned models. To address these shortfalls, this thesis proposes looking into the three stages of supervised learning and devised pre-learning methods, which deal with data and feature selection; in-learning methods, which modify the learning process; and post-learning methods, which modify the prediction process to compensate for shifts in conditions.

The first in-learning method is a transfer learning-based technique that utilizes a limited amount of test data to train further the prediction model pre-trained on general training data. This technique boosted the accuracy of a vocal emotion recognizer by 10%.

For applications that require a timely response, we employed a post-learning strategy in the form of local learning. It handles multiple covariate shifts and improves the prediction accuracy in one vocal emotion recognition instance from 88.8% to 93.2%. Local learning also allows the use of feature augmentation to convert a more difficult concept-shift problem into an easier covariate-shift problem. The resulting controller outperforms PID controllers in water shooting control.

When data are abundant, we leverage pre-learning methods such as condition-specific learning, to avoid non-stationary conditions altogether. Using this technique, we developed a semi-automatic snore labeling software that produces good accuracy (0.93 F1-score) and cuts labeling time from hours to minutes. Alternatively, we use deep learning methods to learn features that are robust to shifts. In our ablation study, we showed that features extracted from very deep networks and recurrent networks results in a more accurate and robust snore classification.

With the advance of computer simulation, unlimited artificial data can be generated to better approximate and cover possible test conditions. We tested this idea in teaching a double-hull welding robot to climb down safely from a high wall through reinforcement learning and achieved a 90% success rate.

Finally, from these applications, we distilled a method selection guideline based on data availability, time urgency, and type of shift.

Contents

Acknowledgements	ix
Abstract	xv
List of Figures	xxiii
List of Tables	xxvii
1 Introduction	1
1.1 Background: Non-Stationary Condition	1
1.2 Literature Review	4
1.3 Thesis Statement	6
1.4 Contributions	7
1.5 Thesis Outline	8
2 Strategy to Compensate for Non-Stationary Conditions	11
2.1 Taxonomy	11
2.1.1 In-Learning	13
2.1.2 Post-Learning	14
2.1.3 Pre-Learning	14
2.2 Selection Guideline	15
2.2.1 Data Availability	16
2.2.2 Time Urgency	16
2.2.3 Shift Type	18
2.2.4 Learning Type	18
I In-Learning Compensation	19
3 Importance Weighted Learning: Selecting Data Implicitly	21
3.1 Concept	21
3.2 Application: Inverse Kinematics Learning	22
3.2.1 Motivation	22
3.2.2 Covariate Shift in Inverse Kinematics Learning	24

3.2.3	Importance Weighted Regression	25
3.2.4	Experiment: 2D Planar Manipulator	25
3.2.5	Discussion	27
3.3	Conclusion	29
4	Transfer Learning:	
	Augmenting Limited Test Data	31
4.1	Concept	31
4.2	Application: Vocal Emotion Recognition	33
4.2.1	Motivation	33
4.2.2	Mixing Speaker-Independent and Speaker-Dependent Approaches	34
4.2.3	Experiment	35
4.2.3.1	Database Selection	35
4.2.3.2	Feature Extraction	36
4.2.3.3	Dataset Division for Each Speaker	38
4.2.3.4	Speaker Combination for SI, SD, and Hybrid Approaches	39
4.2.3.5	Machine Learning Algorithm	41
4.2.3.6	Parameter Optimization and Feature Selection	41
4.2.3.7	Classification Test and Performance Comparison	42
4.2.4	Results	42
4.3	Conclusion	43
II	Post-Learning Compensation	45
5	Local Learning:	
	Combining Nearby Predictions	47
5.1	Motivation for Online Compensation	47
5.2	Concept	49
5.2.1	Local Learning	49
5.2.2	Making IWL Online	51
5.2.3	Approximating Online IWL using Local Learning	53
5.2.4	Multi-Local-Model Learning (MLML)	55
5.3	Application 1: Inverse Kinematic Modeling	57
5.3.1	Method	57
5.3.2	Results and Discussion	58
5.4	Application 2: Vocal Emotion Recognition	58
5.4.1	Label Modification	58
5.4.2	Detecting Covariate Shift	61
5.4.3	Classification Methods	61
5.4.4	Results and Discussion	62
5.5	Conclusion	69

6	Feature Augmentation:	
	Converting Concept Shifts to Covariate Shifts	71
6.1	Concept	71
6.2	Application: Water Shooting Control	74
6.2.1	Motivation	74
6.2.2	Literature Review	74
6.2.2.1	Learning-based Control	75
6.2.2.2	Time-Delay Control	75
6.2.3	Model-based Learning Controller	76
6.2.3.1	Water-Shooting Model	77
6.2.3.2	Feature Augmentation for Non-Stationary Condition	78
6.2.3.3	Online Regression Algorithm	79
6.2.3.4	Prediction Model to Address Time Delay	80
6.2.3.5	Time Delay Estimation to Map Delayed Feedbacks	81
6.2.3.6	Control Policy	83
6.2.4	Experiments	85
6.2.4.1	Computer Simulation Model	85
6.2.4.2	Time Delay Matching	88
6.2.4.3	User Interface for Human Operators	88
6.2.4.4	Model-based Learning Controller vs. Human Op- erators	89
6.2.4.5	Model-based Learning Controller vs. PID Controller	90
6.2.5	Result and Discussion	90
6.2.5.1	Model-based Learning Controller vs. Human Op- erators	90
6.2.5.2	Model-based Learning Controller vs. PID Controller	91
6.2.5.3	Discussion	92
6.3	Conclusion	94

III Pre-Learning Compensation 95

7	Condition-Specific Learning:	
	Avoiding Non-Stationary Condition	97
7.1	Concept	97
7.2	Application: Semi-automatic Snore Labeling	98
7.2.1	Motivation	98
7.2.2	Related Works	99
7.2.3	Overview	100
7.2.4	Snore Labeling Algorithm	101
7.2.4.1	Signal Segmentation	101
7.2.4.2	Uniform Sampling	104
7.2.4.3	Feature Extraction	106
7.2.4.4	Hierarchical Clustering	108

7.2.4.5	Human Labeling	109
7.2.4.6	Classification	110
7.2.5	Experiments	110
7.2.5.1	CVDR Dataset	111
7.2.5.2	GUI	111
7.2.5.3	Verification	113
7.2.5.4	Results	113
7.3	Conclusion	116
8	Deep Learning:	
	Learning Robust Features	119
8.1	Concept	119
8.2	Application: Audio-based Snore Detection	120
8.2.1	Motivation	120
8.2.2	Methods	121
8.2.2.1	Data	122
8.2.2.2	Feature Learning Models	122
8.2.2.3	Leave-One-Out Model Evaluation	126
8.2.2.4	Performance Metrics	126
8.2.3	Results	127
8.2.4	Discussion	130
8.2.4.1	Effect of the Classifier Choice	130
8.2.4.2	Handcrafted vs. Learned Features	131
8.2.4.3	Temporal vs. Spectral Features	131
8.2.4.4	Short-term vs. Long-term Features	132
8.2.4.5	Shallow vs. Deep Networks	132
8.3	Conclusion	133
9	Artificial Data:	
	Generating Your Own Data	135
9.1	Motivation	135
9.1.1	Challenges in Shipbuilding Automation	135
9.1.2	Double-Hull Blocks Welding Robot	137
9.1.2.1	Conceptual Design	137
9.1.2.2	Full-Size Prototype	137
9.1.2.3	Software	140
9.1.2.4	Robot Test	140
9.1.3	Bilge Block Climbing: How to Land from High Walls	142
9.1.4	Simulator: Safer Environment	143
9.1.5	Reinforcement Learning: Learning to Land Safely	143
9.2	Artificial Data Generation	145
9.3	Experiment	146
9.3.1	Setup	146
9.3.1.1	Simulation Environment	146

9.3.1.2 Reinforcement Learning Setup	147
9.3.2 Results and Discussion	150
9.4 Conclusions	153
10 Conclusions	155
10.1 Summary of Contribution	155
10.2 Future Work	158
List of Author's Publications	161
Bibliography	163

List of Figures

2.1	Taxonomy of non-stationary condition compensation techniques . . .	12
2.2	Compensation method selection guideline.	17
3.1	The density ratio of test feature distribution (p_{te}) to training feature distribution (p_{tr}) is used as the importance weights (IW).	22
3.2	Prediction beyond data (covariate shift)	24
3.3	Covariate shift in inverse kinematics of 2D planar manipulator . . .	27
3.4	Comparison between unweighted and weighted inverse kinematics prediction using mean error and variance contour plots.	28
4.1	(a) An audio signal of a sample emotional speech; Features: (b) Voice probability (segments with probability more than 0.5 are considered voiced parts); (c) Pitch contour of the audio signal with blue portions indicating voiced parts, while red indicating otherwise; (d) Energy profile of the audio signal; (e) Speech pause time and speech duration; (f) Profiles for the first four Mel-Frequency Cepstral Coefficients (MFCCs).	37
4.2	Average test accuracy for different proportion of SD training data (SI: 0%; Hybrid: 0% < SD < 100%; SD: 100%)	43
5.1	Results of inverse kinematics prediction with and without covariate shift compensation	59
5.2	Erroneous prediction by LWKLS when the covariate shift is large: (a) Large covariate shift: Test task space (red) is further away from the training task space (blue); (b) AIWKLS: Inaccurate but consistent; (c) LWKLS: Inaccurate and inconsistent	60
6.1	Illustration of feature augmentation	73
6.2	Overview of the proposed model-based learning controller (MBLC)	76
6.3	Water shooting simulation	77
6.4	Finite state machine for exploration and quick initial prediction	84
6.5	Building Fire simulation	86
6.6	Oil rig fire simulation	86
6.7	Non-stationary conditions: (a) Wind gust change as the discrete non-stationary conditions; (b) Rocking boat as the continuous non-stationary conditions.	89

6.8	Comparison between human and model-based learning controllers under sudden wind change and continuous boat motion at two locations.	91
6.9	Comparison between PID and model-based learning controllers under sudden wind change and continuous boat motion at two locations	92
6.10	How model-based learning controller react to change	93
6.11	Improvement due to the prediction model: (a) Error reduction (b) pitch angle change	93
7.1	(a) Input: whole night of snore recording; (b) Output: labeled snore signal; (c) Snore signal (zoomed in); (d) Labeled snore signal (zoomed in)	102
7.2	Six-step semi-automatic snore labeling algorithm	103
7.3	V-Box segmentation process (from top to bottom): input signal; count per box; change markers; merged episodes and segment centers marked by red lines	104
7.4	Feature extraction: input raw signal, energy profile, and power spectrum with prominent peak frequencies for a snore segment (first column) and a noise segment (second column)	105
7.5	Hierarchical clustering of piezoelectric sensor signal segments	107
7.6	GUI of human assisted snore labeling software: (a) PSG data loading; (b) Clustering; (c) Labeling; (d) Classification; (e) Verification; and (f) Results display	112
7.7	Classified piezoelectric signal segments for each subject	114
7.8	Processing time of semi-auto snore labeling	116
8.1	An non-intrusive (pressure and audio sensing), ambulatory sleep monitoring device offers comfortable and long term monitoring at home	122
8.2	Five-second snore segment as a 1D waveform (Top) and its corresponding 2D Mel-spectrogram. The resonant formant frequencies of the snore are clearly shown as the layered texture in Mel-spectrogram but hard to see in the 1D waveform.	123
8.3	Illustration of the model ablation study for snore classification: Each face of the cuboid represent different comparison matrices, namely, classifier and features (front), deep learning architectures (top), different 2D CNN features (side). The five arrows around the cuboid show the axes of comparison.	124
8.4	Leave-one-out model evaluation results of seven snore classification models: The three groups of box plots show the statistics of the precision, recall, and f1-score from 16 subjects. The mean values are marked by 'x' in the plot, while the median by the horizontal lines within the boxes. Both ends of the boxes represent the first and third quartile of the metrics. The whiskers show the minimum and maximum with the maximal length at 1.5 times the inter-quartile range (IQR). Any point outside the whiskers is considered an outlier.	128

9.1	Shipbuilding by assembling double-hull blocks.	136
9.2	Conceptual design	138
9.3	Full size prototype	139
9.4	Testing the mobility and welding function of the full-size welding robot on a cutout block.	141
9.5	Full-size prototype scaling a high wall.	142
9.6	Welding robot simulator scaling a high wall.	144
9.7	A cutout section from a bilge block was used as a simulation environment for the welding robot simulator.	147
9.8	Actions and states in our reinforcement learning formulation: (a) shows the available joint action for action selection; (a) and (b) form the states of the robot.	148
9.9	Evaluation results of reinforcement learning models trained with wide and narrow initial condition range using fixed initial condition with wide and narrow range.	151
9.10	Snapshots of evaluating landing control model learned from reinforcement learning.	153

List of Tables

4.1	Detailed feature set (mean, variance, first and second difference) of each feature group	38
4.2	General variation of vocal features with emotion [1, 2]. Symbols: (‘>’: increase; ‘<’: decrease; ‘=’: no change; ‘ μ ’: mean; ‘ σ ’: standard deviation; ‘ $E(\ )$ ’: energy; ‘ f_H ’: high frequency; ‘ f_M ’: medium frequency; ‘-’: not applicable)	38
4.3	Dataset division for SI approach. For fair comparison, only partial of the training data are used so that it matches the amount of data in other approaches. (Sp: Speaker; Tr: Training Set; CV: Cross-Validation Set; T: Test Set)	39
4.4	Dataset division for SD approach. (Sp: Speaker; Tr: Training Set; CV: Cross-Validation Set; T: Test Set)	40
4.5	Dataset division for Hybrid approach. (Sp: Speaker; Tr: Training Set; CV: Cross-Validation Set; T: Test Set)	40
4.6	Selection of vocal features and regularization parameter	42
5.1	Arousal emotion classification of the EmoDB dataset using logistic regression (LR) and the KS-Test failure rate: Subject 10, 11, 12, and 16 (in bold) have higher-than-average KS-Test failure rates and below-average classification accuracy.	62
5.2	Valence emotion classification of the EmoDB dataset using logistic regression (LR) and the KS-Test failure rate: Subject 12, 13, 14, 15, and 16 (in bold) have higher-than-average KS-Test failure rates and below-average classification accuracy.	63
5.3	Four-basic-emotion classification of the EmoDB dataset using logistic regression (LR) and the KS-Test failure rate: Subject 3, 11, and 16 (in bold) have higher-than-average KS-Test failure rates and below-average classification accuracy.	63
5.4	Arousal classification of EmoDB subjects that suffered from covariate shifts using logistic regression (LR), importance-weighted logistic regression (IWLR), and SOM-based logistic regression (SOMLR): Methods with the highest accuracy for each subject are highlighted in bold. SOMLR improves the performance of all affected subjects but IWLR does not.	65

5.5	Valence classification of EmoDB subjects that suffered from covariate shifts using logistic regression (LR), importance-weighted logistic regression (IWLR), and SOM-based logistic regression (SOMLR): Methods with the highest accuracy for each subject are highlighted in bold. Both SOMLR and IWLR improve or maintain classification performance but SOMLR performs the best.	65
5.6	Four-basic-emotion classification of EmoDB subjects that suffered from covariate shifts using logistic regression (LR), importance-weighted logistic regression (IWLR), and SOM-based logistic regression (SOMLR): Methods with the highest accuracy for each subject are highlighted in bold. SOMLR generally improves the performance except for Subject 16. IWLR only improves the performance of Subject 3. . .	66
5.7	Arousal, valence, and four-basic-emotion classification of the SAVEE dataset using logistic regression and the KS-Test failure rate: Subject 3 and 4 (in bold) have higher-than-average KS-Test failure rates and below-average classification accuracy.	66
5.8	Arousal, valence, and four-basic-emotion classification of SAVEE subjects that suffered from covariate shifts using logistic regression, importance-weighted logistic regression, and SOM-based logistic regression: Methods with the highest accuracy for each subject are highlighted in bold. On average, both SOMLR and IWLR improve the classification results. IWLR performs the best and SOMLR's performance reduces for Subject 4 in the four-basic-emotion classification.	66
5.9	Arousal classification of EmoDB subjects that did not suffer from covariate shifts using logistic regression, importance-weighted logistic regression, and SOM-based logistic regression: Methods with the highest accuracy for each subject are highlighted in bold. IWLR has the highest mean accuracy.	67
5.10	Valence classification of EmoDB subjects that did not suffer from covariate shifts using logistic regression, importance-weighted logistic regression, and SOM-based logistic regression: Methods with the highest accuracy for each subject are highlighted in bold. SOMLR has the best classification performance on average.	68
5.11	Four-basic-emotion classification of EmoDB subjects that did not suffer from covariate shifts using logistic regression, importance-weighted logistic regression, and SOM-based logistic regression: Methods with the highest accuracy for each subject are highlighted in bold. LR has the best classification performance.	68

5.12	Arousal, valence, and four-basic-emotion classification of SAVEE subjects that suffered from covariate shifts using logistic regression, importance-weighted logistic regression, and SOM-based logistic regression: Methods with the highest accuracy for each subject are highlighted in bold. On average, both SOMLR and IWLR improve the classification results. There is no consistent winner that performs the best in all cases.	69
6.1	Simulation Model Specifications	87
6.2	Parameters for Time Delay Estimation	88
6.3	Man vs. Machine Experiment Details	90
7.1	Average and standard deviation of the extrapolated classification metrics	115
8.1	Neural network structure of deep learning models used for feature learning. Notation: Conv(number of kernel, kernel size, stride, padding); MaxPool(kernel size, stride, padding); Linear(Number of output); BatchNorm( ); Dropout(dropout probability); LSTM(input size, hidden state size)	125
8.2	Hyper-parameters used to train each snore classification model. . .	126
8.3	Ablation study to see the incremental effect of different features and classifiers (NN: Neural Networks, SVM: Support Vector Machine): The values in brackets show the differences compared to the VGG SVM, which have the best mean, median, standard deviation, and inter-quartile range (IQR) of the F1-score.	129
9.1	Range of initial condition for welding robot climbing-down simulation.	150
9.2	DQN parameters.	151

Chapter 1

Introduction

This chapter describes the motivation and theme that underlie the research and define the work scope. The first section introduces the main problem the thesis addresses, the non-stationary condition. In the second section, a literature review is done to survey various methods reported by others to address this issue. The last few sections describe the thesis statement, the contribution, and the thesis outline.

1.1 Background: Non-Stationary Condition

Machine learning is attractive for engineers because it has the potential to automate certain engineering design process by just learning from data. For example, controllers can now be constructed from data instead of manually designed using engineering theories [3]. It is also the go-to method when the problem is too complicated to be represented by conventional mathematical models, such as motion control for robots with high degree-of-freedom [4].

However, the success of machine learning techniques relies on data quality. For instance, if a dataset contains wrong labels, the resulting trained model will give wrong predictions. Also, a high-quality dataset should contain sufficient information to generalize well to future unseen situations. This can be achieved by increasing the number of data and also its variety. This idea is exemplified in the ever-increasing number of classes and data points in the ImageNet dataset [5].

Insufficient information in a dataset, either due to poor quality or small quantity, compromises the performance of machine learning techniques.

Besides collecting comprehensive data, is there another way to compensate for the adverse effect of insufficient information or poor quality data on machine learning techniques? This thesis attempts to partially answer this question by looking at non-stationary conditions, where there is a difference in the information contained in a training dataset and information required during deployment. As a start, we will establish the theoretical background describing non-stationary conditions in the remaining of this section.

In statistical machine learning, a classifier or regressor, g , takes an input, $\mathbf{x} \in \mathbb{R}^n$, which can be discrete or continuous, and returns an output (or label), y (discrete for classification or continuous for regression):

$$y = g(\mathbf{x}), \quad (1.1)$$

by assuming both $\mathbf{x}$ and y follow a certain probabilistic distribution:

$$p(\mathbf{x}, y). \quad (1.2)$$

Note that $\mathbf{x}$ is called different names in different fields. For instance, it is called an “input” or a “covariate” in machine learning and statistics. Researchers working on time-series prediction refer to it as a “predictor”, while scholars in pattern recognition use “feature”. In this thesis, we use the terms, “features” and “inputs”, interchangeably. Similarly, we refer y as either “labels” or “outputs”.

With this assumption, learning is done by searching for a function, f , that minimizes the expected prediction error [6]:

$$\min_f \int \int l(f(\mathbf{x}) - y) p(\mathbf{x}, y) d\mathbf{x} dy, \quad (1.3)$$

where the loss function, $l(\cdot)$, is a measurement metric of the error, $e = f(\mathbf{x}) - y$. To work with a real-world finite amount of training data (finite n), we use the following empirical form:

$$\min_f \frac{1}{n} \sum_{i=1}^n l(f(\mathbf{x}_i) - y_i). \quad (1.4)$$

In this formulation, we implicitly assume that the training data is representative of the unseen future test data we want to predict. In other words, we assume that the distribution of training data is equal to the distribution of target test data:

$$p(\mathbf{x}, y)_{tr} = p(\mathbf{x}, y)_{te}. \quad (1.5)$$

However, this assumption is not always valid. The situation in which Equation 1.5 is not valid is called a non-stationary condition [7], concept drift [8], or dataset shift [9]. For consistency, “non-stationary condition” is used throughout the thesis. Under a non-stationary condition, classification or regression accuracy often degrades significantly.

The joint distribution in Equation 1.5 can be further decomposed into a conditional distribution, $p(y|\mathbf{x})$ and an input distribution, $p(\mathbf{x})$:

$$p(\mathbf{x}, y) = p(y|\mathbf{x}) p(\mathbf{x}). \quad (1.6)$$

Therefore, we can change the joint distribution by varying the conditional or the input distribution. A change in the input distribution is termed as a covariate shift [10] or virtual concept drift [8], while a change in the conditional distribution is called a concept shift [10] or real concept drift [8]. For uniformity, the terms “covariate shift” and “concept shift” are used in this thesis.

A covariate shift is defined as a condition in machine learning where the distribution of $\mathbf{x}$ changes between the training and test stage:

$$p(\mathbf{x})_{tr} \neq p(\mathbf{x})_{te}; \quad (1.7)$$

while the relationship between $\mathbf{x}$ and y is constant:

$$p(y|\mathbf{x})_{tr} = p(y|\mathbf{x})_{te}. \quad (1.8)$$

On the other hand, a concept shift is the opposite where the relationship between $\mathbf{x}$ and y changes:

$$p(y|\mathbf{x})_{tr} \neq p(y|\mathbf{x})_{te}; \quad (1.9)$$

while the input distribution is fixed:

$$p(\mathbf{x})_{tr} = p(\mathbf{x})_{te}. \quad (1.10)$$

As compared to a covariate-shift problem, a concept shift problem is harder to solve because the shift cannot be detected without looking at both the input and output data distributions [8]. Consequently, in the context of online learning, we need to wait for the arrival of the unknown output variable to determine if the current prediction model is outdated and requires adaptation [11]. Furthermore, a concept shift can alter the decision boundary significantly, which may render the previously learned model useless. Thus, it is beneficial to convert a real concept-drift problem into a virtual concept-drift problem. This conversion is employed in Chapter 6.

Covariate shifts occur in many real-life applications. For example, in speech recognition, the speech recognition software is usually trained using data from a group of speakers. When the trained system is deployed, the recognition performance often drops because it is tested on new speakers, which might have a slightly different input distribution [12].

1.2 Literature Review

To address covariate shift problems, a class of algorithms called importance weighted learning (IWL) was developed [13]. These algorithms have been shown to improve classification performance in applications such as vocal emotion recognition [12], speaker identification [14], brain-computer interface (BCI) [15], and tumor diagnosis using DNA microarrays [16]. IWL introduces an importance weight W_I [17] into the risk minimization learning criterion:

$$\min_{\hat{f} \in \mathcal{F}} \int \int l(\hat{f}(\mathbf{x}) - y) W_I(\mathbf{x}) p(\mathbf{x}, y)_{tr} d\mathbf{x} dy, \quad (1.11)$$

where l is a loss function, and the model $\hat{f}$ is selected from a class of models, $\mathcal{F}$. The importance weight (IW) is defined as the density ratio between training inputs and test inputs [15]:

$$W_I(\mathbf{x}) = \frac{p(\mathbf{x})_{te}}{p(\mathbf{x})_{tr}}. \quad (1.12)$$

By scaling the risk minimization learning criterion in Equation 1.11, the learning is done as if test data are used in training, even though only training data are used. We can see this by decomposing the joint probability distribution in the risk

minimization formulation:

$$\begin{aligned} \min_{\hat{f} \in \mathcal{F}} \int \int l(\hat{f}(\mathbf{x}) - y) W_I(\mathbf{x}) p(\mathbf{x})_{tr} p(y|\mathbf{x}) p(\mathbf{x})_{tr} d\mathbf{x} dy \\ = \min_{\hat{f} \in \mathcal{F}} \int \int l(\hat{f}(\mathbf{x}) - y) p(y|\mathbf{x}) p(\mathbf{x})_{te} d\mathbf{x} dy. \end{aligned} \quad (1.13)$$

We can see from Equation 1.13 that the training input $p(\mathbf{x})_{tr}$ has been replaced by test input $p(\mathbf{x})_{te}$. Using this weighted criterion, we are minimizing the unbiased estimation of the generalization error even under a covariate-shift condition [15]. This weighting helps to improve the prediction accuracy when covariate shifts occur.

There are a few ways to estimate the IW: kernel density estimation (KDE), which estimates the input density of training and test data separately and finds their ratio [13]; kernel mean matching (KMM), which finds the ratio by matching the statistical moment between the test and training input distribution [16]; logistic regression (LR), which estimates the density ratio using a probabilistic classifier [18]; Kullback-Leibler importance estimation procedure (KLIEP), which estimates the density ratio by minimizing the KL-divergence between training and test input distributions [19]; and unconstrained least squared importance fitting (uLSIF), which fits a regression model to the true density ratio function [20]. Of all the different methods, uLSIF has the fastest computation speed [13].

There are two assumptions for IWL to work. The first assumption is that the distribution support for training data must cover the support for test data [21] to avoid division by zero in the density ratio calculation. Another assumption is that the learning model is mis-specified [17]. In other words, the model space used in the machine learning criterion (see Equation 1.11) does not contain the true model. As a result, the models trained separately by both training data and test data differ. This result is due to the difference in the learning criterion between the original criterion (Equation 1.4) and the weighted criterion (Equation 1.11). If there exists an $\hat{f}$ that minimizes both criteria (the true model, f , is one such solution), the difference in the input distribution does not matter. Under this condition, IWL methods do not improve the prediction accuracy.

While IWL is used mainly in batch learning, methods to compensate for concept shifts are often found in online learning. There are two popular classes of methods

to address concept shift: active methods and passive methods [11]. Active methods first detect if a drift or shift occurs. If a drift is detected, the existing prediction model is discarded, and a new model is learned based on the most recent data. For example, [22] used the Page-Hinkley (PH) Test for drift detection and performed learning by using recent data from a sliding window.

In contrast, passive methods continuously update the model, regardless of whether a drift is detected or not. Most of the passive techniques involve the use of ensemble learning. These methods create an ensemble of prediction models and combine the predictions from all the models into a final prediction. The adaptivity comes from the dynamic combination rules of predictions based on the prediction performance. A representative work in this class is the Dynamic Weighted Majority (DWM) algorithm [23].

While active methods are wasteful in discarding the previously acquired model, passive methods require the learning of multiple models and how to combine their decisions, which is more complicated. These lead us to an alternative class of methods that makes use of contextual information by adding extra information as part of the predictors or input variables [11], such as past model performances [24]. These methods are relatively simpler compared to passive methods, and unlike active methods, they preserve the previously learned models. Chapter 6 proposes a method that makes use of this concept.

1.3 Thesis Statement

A non-stationary condition occurs when there is a mismatch in information learned by a machine learning algorithm and the information needed during deployment. It is a common issue affecting applications ranging from pattern recognition to learning-based control. Traditional methods such as importance weighted learning or forget-and-relearn methods are inadequate when the shift is frequent, and the change requires a timely response. Hence, we proposed taking a broader view of supervised learning and tackle the problem before, during, and after the learning stage. With this new perspective, we developed a transfer-learning method based on the in-learning concept, local-learning method and feature augmentation based on the post-learning concept, condition-specific learning, robust feature extraction

through deep learning, and artificial data generation based on the pre-learning concept. The new taxonomy also gives us a guideline for compensation method selection depending on data availability, time urgency, and type of change. Finally, these methods were verified by applying them in vocal emotion recognition, snore detection, water shooting control under changing winds, and controlling a robot climbing down from a tall wall.

1.4 Contributions

By working on the development of a sleep monitoring system, vocal emotion recognition for social robots, water shooting control under changing wind, and double-hull welding robot, this thesis makes the following contributions:

1. A new and unified view on how to address non-stationary conditions:
  - (a) Expanding the available tools to counter non-stationary conditions by looking at and modifying the process before (condition-specific learning, deep feature learning, and artificial data generation), during (importance-weighted learning and transfer learning), and after (local learning and feature augmentation) the learning stage of supervised learning.
  - (b) A taxonomy to classify the compensation methods into pre-learning, in-learning, and post-learning, which serves as a guideline to select a compensation method according to data availability, time urgency, and shift type.
2. Application in vocal emotion recognition:
  - (a) Minimizing the drop in performance in vocal emotion recognition due to subject variation by further training using limited data from the test subject.
  - (b) Identifying human subjects suffering covariate shift in vocal emotion recognition and improving the accuracy using local learning.
3. Application in snore detection:

- (a) Semi-automatic snore labeling that allows the labeling of one whole night of snore recording by using only a few samples labeled by human scorers, using condition-specific learning to avoid non-stationary conditions altogether.
- (b) Improving both accuracy and robustness of snore detection across different subjects by learning robust features through a very deep convolutional neural network or recurrent neural network.

4. Application in control:

- (a) Designing a predictive controller to compensate for delays and disturbances caused by wind and vehicle motion for water shooting over a long distance, using local learning to convert a difficult concept-shift problem to a simpler covariate-shift problem.
- (b) Developing the capability of the welding robot to climb down safely from a high wall by using simulations to generate data from different scenarios for reinforcement learning.

1.5 Thesis Outline

This thesis explores different ways to address non-stationary conditions occurring in engineering applications. Chapter 1 provides the background, motivation, and theoretical modeling of the problem. It also reviews a few techniques used to compensate for shifts in condition.

However, the techniques reported are inadequate. They are not able to handle multiple shifts and provides timely responses. To develop new techniques, a taxonomy is devised in Chapter 2 based on the theory described in Section 1.1. The taxonomy classifies the compensation methods into three groups: pre-learning, in-learning, and post-learning, depending on which stage the method modifies. The taxonomy also serves as a guideline on how to approach problems involving non-stationary conditions.

The rest of the thesis is divided into four parts. In Part I, in-learning methods are explored. Chapter 3 tests conventional importance-weighted learning using a toy inverse kinematics problem and discusses the shortfalls of the method. Transfer

learning, another in-learning method, is employed in Chapter 4 to improve vocal emotion recognition performance.

The post-learning methods are discussed in Part II. Local learning in Chapter 5 is applied to improve the accuracy of vocal emotion recognition in subjects identified as suffering from covariate shift. The method is also used in conjunction with feature augmentation to turn a concept-shift problem into a covariate-shift problem for water shooting control in Chapter 6. The design and the verification of the water shooting controller were also published in [25].

Part III covers the pre-learning methods, such as condition-specific learning, used in Chapter 7 for semi-automatic snore labeling (also published in [26]), and deep learning, used in Chapter 8 for features robust to subject variation in snore detection. The results of extracting robust features through deep learning methods was first presented in [27]. Lastly, Chapter 9 explores the use of deep reinforcement learning and simulation to teach the robot to climb down safely from a high wall. The use of computer simulation gives us the freedom to generate desired training data that cover the deployment condition.

Finally, we conclude the thesis and discuss future work in Chapter 10.

Chapter 2

Strategy to Compensate for Non-Stationary Conditions

In Chapter 1, all the compensation methods reviewed try to model the changing condition using the probabilistic model shown in Equation 1.6. This thesis expands the repertoire of compensation techniques by looking at how to utilize the probabilistic model at different stages of the supervised learning process. In fact, we can group all the compensation techniques according to the stages of machine learning. This chapter describes the taxonomy developed to classify the compensation techniques. This taxonomy can serve as a guideline for compensation techniques selection.

2.1 Taxonomy

Supervised learning can be divided into three stages: pre-learning, in-learning, and post-learning. They correspond to the following components: training data, learning algorithm, and learned prediction model, respectively, in the supervised learning process, as shown in Figure 2.1. In this simplified process flow, learning algorithms trained on training data with labels, $(\mathbf{X}_{tr}, Y_{tr})$, to generate a prediction model, $f(\mathbf{x})$, that can anticipate labels, $\mathbf{Y}_p$, for unlabeled test data, $\mathbf{X}_{tr}$. Note that in this thesis, we generally use training data to refer to data used only for training, while data generated from the same condition as the deployment or test

FIGURE 2.1: Taxonomy of non-stationary condition compensation techniques

condition are also referred to as test data. These test data may be available before deployment and hence can be used in training.

At any of these stages, it is possible to modify the process in such a way that the final prediction is adapted to the test condition to address the non-stationary condition. Hence, we can group compensation techniques according to the three stages. Figure 2.1 also lists the techniques used in this thesis according to this new grouping. The following subsection describes each of these groups in detail.

2.1.1 In-Learning

At the in-learning stage, machine learning algorithms process data to construct a prediction model. We can modify the learning process such that the prediction model produced is accustomed to the test condition. This result can be achieved by having the learning algorithm learn more from data that are generated from the same condition as the test condition. This is akin to the data selection method mentioned in Sub-Section 2.1.3, but having the selection done implicitly by giving more influence to certain data during the learning process.

In this class of methods, no change or modification to the training data or prediction process is necessary. Prior knowledge of the test condition, usually acquired from features of test data, is the only requirement.

Importance-Weighted Learning (IWL) discussed in Chapter 1 is an example of this type of compensation technique. It uses importance weighting to give more weight to data that are relevant to the test condition during the learning process. Chapter 3 shows an example application of this method in kinematic model learning.

Alternatively, if some data (both feature and labels) from the test condition are available, we can perform relearning to combine the information from both the new test data and the original training data. This idea can be implemented through transfer learning, as we shall see in Chapter 4.

Another example technique used in online learning is forgetting, which we mentioned in Section 1.2. It discards the old model and relearns a new model once it has detected a condition change.

2.1.2 Post-Learning

Even when the data are fixed, and the prediction model is already learned, compensation can still be done to address the non-stationary condition. This class of methods alters the prediction process to adapt to the change in condition. However, this is only limited to methods that allow dynamic prediction, where prediction can change according to certain criteria. Since there is no special data preparation or modification to the learning algorithm, this class of methods does not require prior knowledge of the test condition.

An instance of this class of methods is local learning. It allows dynamic prediction by learning multiple models. Consequently, it can select predictions from the most suitable model during the prediction stage. We shall discuss this method more in Chapter 5. Ensemble learning, discussed in the literature review section in Chapter 1, also employs a similar strategy.

Another example described in Chapter 6 uses an expanded input space (feature augmentation) to embed different models. As a result, dynamic prediction can be achieved easily by simply supplying extra inputs during the prediction evaluation.

2.1.3 Pre-Learning

In the pre-learning stage, the main processes are data preparation and feature extraction. We can minimize the effect of a non-stationary condition, by clever data preparation or better feature extraction alone, without modifying the learning process or changing the way a prediction is made. The main idea is to make sure the training data are similar to the test data:

$$p(\mathbf{x}, y)_{tr} \approx p(\mathbf{x}, y)_{te}. \quad (2.1)$$

For data preparation, one can ideally amass sufficient data that cover every possible situation. Learning on this type of data makes sure that part of the training data cover the test condition, and hence the non-stationary condition does not occur. Practically, it is always challenging to collect such comprehensive data. It requires full knowledge of the population or process that generates the data. Without this information, we cannot be certain that the dataset is representative of the data

population. To alleviate this uncertainty, a large amount of data is needed, which only big institutions have the resource to collect. For others, an alternative way is needed. One solution is to exploit the advance of computational modeling and simulation to generate a massive amount of data through simulation (artificial data). Chapter 9 explores this direction.

Alternatively, if we have prior knowledge of the test condition, we can limit the training data to only those generated from the same condition (condition-specific), provided there are sufficient training data of the same condition. In this way, the model learned will be attuned to the test condition, and the prediction performance will improve. We shall see the implementation of this idea in Chapter 7.

For feature extraction, we wish to extract features that are invariant or robust to condition change. This will minimize the performance drop due to condition change. Painstaking feature engineering is required to extract invariant features, but the achievement is often limited, especially when the feature dimension is high [28]. The use of feature-dimension reduction is possible, but the resulting features often lose the original physical meaning, making explaining the results difficult [29, 30]. Another way is to generate many features and perform feature selection based on correlation or other information criteria [31], which is equally tedious. Fortunately, recent studies show that extracting hierarchical features corresponding to different levels of abstraction [32] is a better way to perform feature extraction. To this end, Chapter 8 uses deep learning techniques to extract features that are more robust to condition change.

2.2 Selection Guideline

Grouping the compensation techniques according to this taxonomy is useful. It allows us to understand the characteristics of each class of methods better and subsequently helps us to choose a suitable compensation method for our application.

As mentioned in the previous section, both the pre- and in-learning methods require prior knowledge of the test condition. The test condition is usually inferred from the test data, suggesting a certain amount of test data is necessary. On the other hand, the post-learning methods have no such requirement. These methods learn

multiple models, making them more flexible. If prior knowledge is not available, other pre-learning methods require numerous data.

From these characteristics, we can deduce the selection guideline to help us to choose the right method for our application. We can devise the following criteria: data availability, time urgency, change frequency, shift type, and learning type. Figure 2.2 summarizes the selection guideline.

2.2.1 Data Availability

If a great amount of training data is available beforehand, we can consider pre-learning methods. In Chapter 7, a sufficient number of snore data allowed us to construct an individual classifier for each subject (condition-specific data). Similarly, the application of deep learning methods in snore detection (Chapter 8) is possible because of the abundance of data.

For methods that require prior knowledge, a few test data are necessary before and during the learning process to infer the test condition. This requirement applies to in-learning methods, such as importance-weighted learning and transfer learning. These techniques are applied in manipulator kinematics modeling in Chapter 3 and vocal emotion recognition in Chapter 4. In both applications, a certain amount of test data is available.

2.2.2 Time Urgency

As both the pre- and in-learning methods undergo data re-sampling or relearning when the condition changes, predictions cannot be made in time. Hence, they are not suitable for time-critical applications such as robot control and perception. In this situation, post-learning methods are preferred as the adaptive prediction procedure is usually much less expensive than re-learning. This urgency is exemplified in the water shooting control application in Chapter 6, where control decisions need to be made in real-time.

FIGURE 2.2: Compensation method selection guideline.

2.2.3 Shift Type

Since it is more expensive for pre- and in-learning methods to adapt to condition change, applications that involve frequent condition change should employ post-learning methods. Because of this, local learning is a more practical choice for the vocal emotion recognition application (see Chapter 5) since it can adapt to different subjects (conditions).

Another consideration is the type of shift. For a more general type of shift, including the covariate shift and concept shift, pre-learning is the preferred method. As long as the condition change is known beforehand, one can always prepare suitable training data that match the deployment condition (see Chapter 7). Extracting features that are invariant or robust to the condition change (Chapter 8) or using comprehensive data to cover all possible conditions (Chapter 9) are other effective ways under this class of methods. Not all in- and post-learning methods can handle general condition change. Importance-weighted learning in Chapter 3 and local learning in Chapter 5 can only handle covariate shifts. In an online learning setting, both forgetting and predictor augmentation (Chapter 6) can be used to deal with general shifts.

2.2.4 Learning Type

Post-learning methods can be applied either in batch or online since they do not modify the data or learning process. However, pre-learning methods can only work in batch learning. Lastly, in-learning methods can work in either an online or a batch setting, depending on the specific method. For example, the forgetting mechanism mentioned in Section 1.2 is an online-learning method, while importance-weighted learning (see Section 1.2 and Chapter 3) is a batch learning method.

Part I

In-Learning Compensation

Chapter 3

Importance Weighted Learning: Selecting Data Implicitly

Importance weighted learning (IWL) is one of the major techniques used to remedy the covariate shift issue. In this chapter, we test the technique using a toy inverse kinematics problem to appreciate its strengths and weaknesses better. It motivates the development of other shift compensation methods that address the weaknesses of IWL in subsequent chapters.

3.1 Concept

The concept of importance weighting was discussed in Chapter 1. The main idea is to include the ratio of test feature distribution to training feature distribution (an example is shown in Figure 3.1) in the learning equation (Equation 1.11), so that data that are similar in feature space have more influence during learning. This idea is akin to selecting data that are similar to those generated in the test condition, though it is done indirectly through weighting during learning. As a result, the prediction model is attuned to the test condition and can perform well in the test condition.

Also reviewed in Section 1.1 are the different methods used to estimate the importance weights. They include kernel density estimation (KDE), which estimates the input density of training and test data separately and finds their ratio [13]; kernel

FIGURE 3.1: The density ratio of test feature distribution (p_{te}) to training feature distribution (p_{tr}) is used as the importance weights (IW).

mean matching (KMM), which finds the ratio by matching the statistical moment between the test and train input distribution [33]; logistic regression (LR), which estimates the density ratio using a probabilistic classifier [18]; Kullback-Leibler importance estimation procedure (KLIEP), which estimates the density ratio by minimizing the KL-divergence between the training and test input distribution [19]; and unconstrained least squared importance fitting (uLSIF), which fits a regression model to the true density ratio function [13]. Among them, we chose uLSIF (see Algorithm 1) with leave-one-out cross-validation (LOOCV) to estimate the importance weights in the next section due to its fast computation speed.

3.2 Application: Inverse Kinematics Learning

3.2.1 Motivation

In inverse kinematics, we determine the joint angles, $\boldsymbol{\theta}$ of a mechanism given a desired pose of the end effector, $\boldsymbol{x}$:

$$\boldsymbol{\theta} = \boldsymbol{f}^{-1}(\boldsymbol{x}). \quad (3.1)$$

Algorithm 1: uLSIF with LOOCV from [13]

Input: n_{tr} training features, $\{\mathbf{x}_{tr}^{(i)}\}_{i=1}^{n_{tr}}$; n_{te} test features, $\{\mathbf{x}_{te}^{(j)}\}_{j=1}^{n_{te}}$.

Hyperparameters: a set of Gaussian width, σ ; regularization parameter, λ .

Output: importance weight function, $W_I(x)$.

Algorithm:

$t \leftarrow \min(100, n_{te})$.

$n \leftarrow \min(n_{tr}, n_{te})$.

Randomly choose t centers $\{\mathbf{c}_l\}_{l=1}^t$ from $\{\mathbf{x}_{tr}^{(j)}\}_{j=1}^{n_{tr}}$ without replacement.

For each candidate of Gaussian width, σ

$$H_{l,l'} \leftarrow \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \exp \left(-\frac{\|\mathbf{x}_{tr}^{(i)} - \mathbf{c}_l\|^2 + \|\mathbf{x}_{tr}^{(i)} - \mathbf{c}_{l'}\|^2}{2\sigma^2} \right) \text{ for } l, l' = 1, 2, \dots, t.$$

$$h_l \leftarrow \frac{1}{n_{te}} \sum_{i=1}^{n_{te}} \exp \left(-\frac{\|\mathbf{x}_{te}^{(i)} - \mathbf{c}_l\|^2}{2\sigma^2} \right) \text{ for } l = 1, 2, \dots, t.$$

$$X_{tr}^{(l,i)} \leftarrow \exp \left(-\frac{\|\mathbf{x}_{tr}^{(i)} - \mathbf{c}_l\|^2}{2\sigma^2} \right) \text{ for } i = 1, 2, \dots, n \text{ and } l = 1, 2, \dots, t.$$

$$X_{te}^{(l,i)} \leftarrow \exp \left(-\frac{\|\mathbf{x}_{te}^{(i)} - \mathbf{c}_l\|^2}{2\sigma^2} \right) \text{ for } i = 1, 2, \dots, n \text{ and } l = 1, 2, \dots, t.$$

For each candidate of regularization parameter, λ

$$\mathbf{B} \leftarrow \mathbf{H} + \frac{\lambda(n_{tr} - 1)}{n_{tr}} \mathbf{I}_t.$$

$$\mathbf{B}_0 \leftarrow \mathbf{B}^{-1} \mathbf{h} \mathbf{1}_n^T + \mathbf{B}^{-1} \mathbf{X}_{tr} \text{diag} \left(\frac{\mathbf{h}^T \mathbf{B}^{-1} \mathbf{X}_{tr}}{n_{tr} \mathbf{1}_n^T - \mathbf{1}_t^T (\mathbf{X}_{tr} \cdot \mathbf{B}^{-1} \mathbf{X}_{tr})} \right).$$

$$\mathbf{B}_1 \leftarrow \mathbf{B}^{-1} \mathbf{X}_{te} + \mathbf{B}^{-1} \mathbf{X}_{tr} \text{diag} \left(\frac{\mathbf{1}_t^T (\mathbf{X}_{te} \cdot \mathbf{B}^{-1} \mathbf{X}_{tr})}{n_{tr} \mathbf{1}_n^T - \mathbf{1}_t^T (\mathbf{X}_{tr} \cdot \mathbf{B}^{-1} \mathbf{X}_{tr})} \right).$$

$$\mathbf{B}_2 \leftarrow \max \left(\mathbf{O}_{t \times n}, \frac{n_{tr} - 1}{n_{tr}(n_{te} - 1)} (n_{te} \mathbf{B}_0 - \mathbf{B}_1) \right).$$

$$LOOCV(\sigma, \lambda) \leftarrow \frac{\|(\mathbf{X}_{tr} \cdot \mathbf{B}_2)^T \mathbf{1}_t\|^2}{2n} - \frac{\mathbf{1}_t^T (\mathbf{X}_{te} \cdot \mathbf{B}_2) \mathbf{1}_n}{n}.$$

end

end

$$(\sigma', \lambda') \leftarrow \arg \min_{(\sigma, \lambda)} LOOCV(\sigma, \lambda).$$

$$H'_{l,l'} \leftarrow \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \exp \left(-\frac{\|\mathbf{x}_{tr}^{(i)} - \mathbf{c}_l\|^2 + \|\mathbf{x}_{tr}^{(i)} - \mathbf{c}_{l'}\|^2}{2\sigma'^2} \right) \text{ for } l, l' = 1, 2, \dots, t.$$

$$h'_l \leftarrow \frac{1}{n_{te}} \sum_{i=1}^{n_{te}} \exp \left(-\frac{\|\mathbf{x}_{te}^{(i)} - \mathbf{c}_l\|^2}{2\sigma'^2} \right) \text{ for } l = 1, 2, \dots, t.$$

$$\boldsymbol{\alpha} \leftarrow \max(\mathbf{0}_t, (\mathbf{H}' + \lambda' \mathbf{I}_t)^{-1} \mathbf{h}').$$

$$W_I(\mathbf{x}) \leftarrow \sum_{l=1}^t \alpha_l \exp \left(\frac{\|\mathbf{x} - \mathbf{c}_l\|^2}{2\sigma'^2} \right).$$

Inverse kinematics is useful in robot control [34]. Unlike forward kinematics, it is often difficult to obtain analytical solutions for inverse kinematics, especially in the case of hyper-redundant manipulators. Furthermore, inverse kinematics is further complicated by one-to-many mappings. To avoid the complexity of deriving the analytical formulation of inverse kinematics, researchers in the field resort to machine learning-based methods.

By using machine learning algorithms, robots can learn the function approximation (regression) to the inverse models from data [4, 34, 35]. This gives rise to a new issue: data collection. As machine learning methods rely heavily on the quality of data, we should try to collect data that covers all the possible state space. However, given the large state space in robots with a high degree of freedom, it is practically impossible. In most cases, we can only be contented with a limited amount of data that covers a portion of the state space. In this situation, we want to make full use of the limited amount of data and extrapolate the prediction model as far as possible beyond the existing data domain (see Figure 3.2 for an illustration). This problem can be described as a covariate-shift problem.

FIGURE 3.2: Prediction beyond data (covariate shift)

3.2.2 Covariate Shift in Inverse Kinematics Learning

Inverse kinematics problems can be cast into a covariate-shift problem. In inverse kinematics, the mapping $\mathbf{f}^{-1}$ between the end effector pose $\mathbf{x}$ (input) and joint

angles θ (output) for a manipulator is constant. This constant mapping is represented by the constant conditional distribution in Equation 1.8. In learning-based inverse kinematics, a robot or manipulator learns the inverse kinematics model from data generated from a section of the full task space. It then uses the learned inverse model to predict the joint angles needed for the end effector to operate in the neighboring sections of the task space. The change in task space can be described as a change in the input distribution (see Equation 1.7). With the constant inverse mapping and changing input distribution, this exactly matches the definition of a covariate shift. Hence, we can apply the covariate-shift concept to inverse kinematics learning.

In the following sub-section, we explore the use of importance weighted learning (IWL) to compensate for the shift caused by the desire to extend the prediction domain of learning-based inverse kinematics.

3.2.3 Importance Weighted Regression

Since the problem of inverse kinematics is a regression problem, an importance weighted version of a kernel least-squared regression is used. The regression method is called adaptive importance-weighted kernel least-squared (AIWKLS) regression [13], as shown in Algorithm 2. An extra parameter γ and kernel width h are used to regularize the complexity of the learned function to avoid overfitting and stability issues. They can be determined through a leave-one-out-cross-validation (LOOCV) procedure.

3.2.4 Experiment: 2D Planar Manipulator

To prove the concept illustrated in the previous subsection, We used a simple 2D planar manipulator as a test case (Figure 3.3(a)). The inverse kinematics of the manipulator are given below:

$$\theta_2 = \cos^{-1} \left(\frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1a_2} \right); \quad (3.2)$$

$$\theta_1 = \tan^{-1} \left(\frac{y(a_1 + a_2 \cos \theta_2) - x(a_2 \sin \theta_2)}{x(a_1 + a_2 \cos \theta_2) + y(a_2 \sin \theta_2)} \right). \quad (3.3)$$

Algorithm 2: Adaptive Importance-Weighted Kernel Least-Squared Regression (AIWKLS)

Input: training data, $\{\mathbf{x}_{tr}^{(i)}, y_{tr}^{(i)}\}_{i=1}^{n_{tr}}$; test features, $\{\mathbf{x}_{te}^{(j)}\}_{j=1}^{n_{te}}$.

Output: predicted test labels, $\{\hat{y}_{te}^{(j)}\}_{j=1}^{n_{te}}$.

Algorithm:

1. Form Gaussian kernel matrix from training features:

$$K_{i,j} = \exp\left(-\frac{\|\mathbf{x}_{tr}^{(i)} - \mathbf{x}_{tr}^{(j)}\|^2}{2h^2}\right) \text{ for } i = 1, \dots, n_{tr}; j = 1, \dots, n_{te}.$$

2. Form importance weight matrix using uLSIF (Algorithm 1):

$$W_{ii} = W_I\left(\mathbf{x}_{tr}^{(i)}\right) \text{ for } i = 1, \dots, n_{tr}.$$

3. Form regression model parameters vector: $\boldsymbol{\beta} = (\mathbf{K}\mathbf{W}^\gamma\mathbf{K})^{-1}\mathbf{K}\mathbf{W}^\gamma\mathbf{y}_{tr}$.

4. Predict test labels: $\hat{y}_{te}^{(j)} = \boldsymbol{\beta}^T \mathbf{x}_{te}^{(j)}$ for $j = 1, \dots, n_{te}$.
-

Using the analytical solution in Equation 3.2 and 3.3 with link lengths $a_1 = 1$ and $a_2 = 1$, 400 data points normally distributed around the end effector pose at $(x = 1, y = 0.6)$ with a standard deviation of $\sigma = 0.1$ (blue circles in Figure 3.3(b)) were generated. Gaussian noise with a standard deviation of 0.005 was added to the data.

Next, kernel least squared (KLS) regression and AIWKLS (Algorithm 2) were implemented to learn the inverse kinematics model, which is used to predict the joint angles needed to move the end effector to draw circles (red circles in Figure 3.3(b)) in task spaces around the data. Note that for AIWKLS, the learning and prediction were performed together (see Algorithm 2). Hence, the learning and prediction had to be done separately for each red circle shown in Figure 3.3(a). In contrast, KLS only needs to perform learning once, and the resulting prediction model is used for all the red circles immediately.

Subsequently, the difference (mean squared error, MSE) between the predicted joint angles and the joint angles calculated using Equation 3.2 and 3.3 were determined. The whole procedure was repeated five times, and the mean error and variance of the prediction by the two methods were determined. Figure 3.4 shows the contour plots of the mean error and variance across the full task space. From the figures, we can see that the accuracy of the joint angle prediction decreases when the task

FIGURE 3.3: Covariate shift in inverse kinematics of 2D planar manipulator

space to be predicted moves away from the data center at $(x = 1, y = 0.6)$ (covariate shift). If we use the MSE contour of 0.001 as a reference, we can see a larger area bounded by the 0.001-contour in AIWKLS than in KLS. In other words, AIWKLS helps to compensate for the shift and reduce the decrease in prediction accuracy when the prediction domain is extended towards the boundary of the training data.

From Figure 3.4, we also observe that the variance of AIWKLS is larger than KLS. The innermost contours of both methods cover about the same area, but KLS has a smaller contour value (5×10^{-8}), compared to the contour value of AIWKLS (5×10^{-7}). The extra prediction task space of AIWKLS comes with an additional cost of consistency. The increase in variance is probably due to the heavier use of border data points that are smaller in number.

3.2.5 Discussion

This experiment has shown that the covariate-shift concept can be used to describe inverse kinematics model learning. Adaptive Importance Weighted Kernel Least Squared Regression (AIWKLS), which is developed to address covariate shifts, was implemented in the inverse kinematics model learning for a simple 2D planar manipulator. The results show that the method can help not only to improve the learned inverse kinematics model's prediction accuracy, but also the prediction range.

FIGURE 3.4: Comparison between unweighted and weighted inverse kinematics prediction using mean error and variance contour plots.

However, during the implementation of AIWKLS, we also noticed that the method requires prior information in the form of test features in batch during learning. In the experiment, separate learning and prediction are needed for each different red circle. In other words, when the covariate shift changes, for example, when the prediction task space change from one red circle to another, re-learning is required. IWL is a batch compensation method, which is not suitable if we want to have continuous predictions, regardless of changing shifts. It is also a batch learning method, which makes continuous robot control model learning (on-line learning) not possible. Lastly, due to the batch compensation nature, where multiple learning passes are needed, the overall computation time increases.

3.3 Conclusion

In this chapter, we have studied importance weighted learning (IWL) in detail. It is an in-learning method as it modifies the training process to adapt to a condition change. Through the toy example (inverse kinematics modeling of a 2D planar manipulator), we found that the technique is effective to address covariate shifts. However, we also realized that the method is both a batch compensation and a batch learning method. It can neither perform online prediction and adapt to multiple shifts in condition nor carry out continuous learning to expand the task space, both of which are useful for robot control. Also, IWL is designed to address covariate shifts only. The method will not be useful to handle generic shifts such as concept shifts. These shortcomings shall be addressed in Chapter 5 and Chapter 6.

Chapter 4

Transfer Learning: Augmenting Limited Test Data

Importance-Weighted Learning (IWL) in Chapter 3 assumes the existence of prior knowledge of the test condition in the form of test features. What happens if we have access to not only the features but also labels from the test condition? In this chapter, we explore the use of transfer learning to further extract information from a limited amount of test data containing both features and labels.

4.1 Concept

Transfer learning [36] applies the knowledge acquired from a training (source) domain, $\mathcal{D}_{tr}$, and task, $\mathcal{T}_{tr}$:

$$\begin{aligned}\mathcal{D}_{tr} &= \{\mathcal{X}_{tr}, p(\mathbf{x}_{tr})\}, \\ \mathcal{T}_{tr} &= \{\mathcal{Y}_{tr}, p(y_s|\mathbf{x}_{tr})\};\end{aligned}\tag{4.1}$$

in another slightly different test (target) domain, $\mathcal{D}_{te}$, and task, $\mathcal{T}_{te}$:

$$\begin{aligned}\mathcal{D}_{te} &= \{\mathcal{X}_{te}, p(\mathbf{x}_{te})\}, \\ \mathcal{T}_{te} &= \{\mathcal{Y}_{te}, p(y_t|\mathbf{x}_{te})\}.\end{aligned}\tag{4.2}$$

In this setting, $\mathbf{x} \in \mathcal{X}$ is the feature, $p(\mathbf{x})$ is the feature distribution, $y \in \mathcal{Y}$ is the label, and $p(y|\mathbf{x})$ is the conditional distribution. Note that the literature in transfer

learning uses the terms “source” and “target”, which correspond to training data and test data. To be consistent with other chapters in this thesis, we replace “source” and “target” with “training” and “test”, respectively. Transfer learning covers a broader problem spectrum than the non-stationary condition considered in this thesis as it allows differences in domains ($\mathbf{x}_{te} \neq \mathbf{x}_{tr}$) or tasks ($y_{te} \neq y_{tr}$) between training and test.

Depending on the availability of labeled data, transfer learning can be applied in three different settings: (a) inductive transfer learning, (b) transductive transfer learning, and (c) unsupervised transfer learning [37]. Inductive learning assumes the availability of labeled data in a test domain, while transductive assumes labeled data are available only in a training domain. Lastly, unsupervised transfer learning assumes no labeled data in either domain. Using this taxonomy, IWL belongs to transductive transfer learning. We attempt to explore inductive transfer learning in this chapter, as we assume the availability of limited labeled test data.

In terms of approach, transfer learning offers the following: (a) instance transfer, (b) feature-representation transfer, (c) parameter transfer, and (d) relational-knowledge transfer [37]. Instance transfer approaches re-weight labeled data in the training domain for use in the test domain. This is the approach used by IWL. Feature-representation transfer approaches try to find common features, while parameter transfer approaches find shared model parameters that work well on both domains. Finally, relational-knowledge transfer approaches map the relational knowledge between the two domains. It assumes the data are not independent and identically distributed (i.i.d.), and hence there exists a relationship among data that can be transferred between domains.

Three of the transfer learning approaches discussed have been applied using deep learning techniques [38]. Instance-based deep transfer learning follows the same idea as instance transfer approaches, while mapping-based deep transfer learning uses the feature-representation transfer approaches. Applying the concept of shared parameters in parameter transfer approaches, network-based deep transfer learning reuses part of the network from a training domain in a test domain. An example of this approach is shown in [39]. The authors successfully transferred a neural network trained on a large-scale annotated dataset to other visual recognition tasks with a limited amount of training data by reusing mid-layers of the networks.

In this chapter, we attempt a crude form of instance and parameter transfer learning using a fully connected artificial neural network. It first performs learning on a large general dataset (training data). Then, it undergoes re-training using a smaller but specific dataset (test data). In doing so, the classifier learns both common concepts applicable across domains and also specific concepts applicable only in specific domains.

The idea of two-stage training comes from the need to initialize the network parameters to random values before training. Through gradient descent, we move the parameters from a random starting position to, hopefully, the most optimal point in the parameter space. Ideally, this point should be closer to the final optimal point. In our approach, this better starting point is obtained by pre-training with general training data.

The transfer learning is achieved partly through the mixed use of both the training data and the limited test data, and the shared parameters in the network. This method is not a fully instance-transfer approach, since it does not re-weight the training data directly but does so in the second learning stage where test data can modify the model pre-trained by the training data. In other words, the test data have more say in shaping the final prediction model. This proposed method is also not a concrete instance of parameter-based transfer learning. Although the whole network is reused, the network parameters are fine-tuned in the second learning stage.

This transfer learning idea is applied to improve vocal emotion recognition across different speakers in the following section.

4.2 Application: Vocal Emotion Recognition

4.2.1 Motivation

According to the National Institute of Mental Health (NIMH), mood disorders is one of the major mental disorders, affecting 9.5% of the population in the United States [40]. However, the existing mental healthcare system is understaffed to meet the public mental health needs [41]. One of the promising technologies to address

this problem is using social robots to monitor one’s mood [42]. To achieve that, the robot must be equipped with emotion recognition capability.

There are two major channels for robots to recognize emotion: facial expression and vocal prosody. Emotion recognition through facial expressions has the best accuracy, but is sensitive to head orientation. In contrast, recognition through vocal features is omnidirectional. The drawback of vocal emotion recognition is its accuracy. Over the years, mixed results have been produced in the field. The results range from as low as 45% [43] to as high as 90% accuracy [44]. The majority of the results have an average accuracy of around 50%, as seen in [43, 45–47].

One reason for the poor performance is the use of the speaker-independent (SI) approach [48]. In the SI approach, classifiers are trained using a database of a mix of different speakers to classify the emotion of a specific speaker. Though the use of the SI database provides sufficient data for training, the accuracy is usually low.

The performance can be improved with the use of the speaker-dependent (SD) approach [48]. SD classifiers are trained using a single specific speaker to classify the emotion of the same speaker. The difficulty of this approach is to obtain sufficient data from the particular speaker to train a classifier. This renders the SD approach impractical in robotic vocal emotion recognition.

4.2.2 Mixing Speaker-Independent and Speaker-Dependent Approaches

To address the trade-offs between the SI and SD approaches, a hybrid method (Hybrid) is proposed to merge them. In this method, a classifier is pre-trained with SI training data followed by SD training with limited SD training data. Putting this in the context of a robot, the initial SI training is the generic training received by the robot before deployment. The subsequent SD training is similar to on-site training the robot performs when it is assigned to a specific user. In this way, we hope the robotic vocal emotion recognition becomes practical, with less SD training data required and yet with accuracy comparable to SD.

4.2.3 Experiment

This section details the experimental procedures to train and test the SI, SD, and Hybrid vocal emotion classifiers. Generally, to construct a classifier, a database with sufficient samples is needed. Then, features that are relevant for classification are extracted. The database is divided into three groups. The first group is used in machine learning to train the classifier. The parameters used in machine learning are selected based on the classification performance of the second group. Finally, using the last group, the performance of the classifier is measured. Besides this general division, each approach has a different set of speakers for training and validation. This difference in speakers marks the distinction between the three approaches. The performance of the three approaches is compared in two dimensions: accuracy and account of SD training data required.

4.2.3.1 Database Selection

Four criteria are considered in database selection. They are the choice of emotion class, human recognition, neutral speech content, and availability. The first criterion is the inclusion of six basic emotions (fear, sadness, happiness, anger, disgust, and surprise), which is inspired by the concept of universal emotion expression [49]. This criterion makes the classification results using different vocal emotion datasets comparable, as long as they share the six basic emotions. The second criterion assumes that human beings are the best vocal emotion classifiers. If a person cannot recognize emotions in the database, it could be even harder for machines. Hence, the affective content in the datasets must be recognizable by human listeners. The third criterion ensures the affective element of the speech is contained in vocal features instead of the speech context. Lastly, the database should be freely and widely available so that more research activities can be done using the same databases. This criterion results in more comparable results.

Using these criteria, two databases were selected: Surrey Audio-Visual Expressed Emotion (SAVEE) Database¹ [50] and Berlin Emotional Speech Database (EmoDB)² [51]. The former features 360 British English utterances by three speakers with

---

¹<http://kahlan.eps.surrey.ac.uk/savee/>

²<http://emodb.bilderbar.info/docu/>

the six basic emotions plus a neutral emotion. The later features 535 German utterances by ten speakers. It contains the basic emotions except for surprise, which is replaced by boredom, plus a neutral emotion.

4.2.3.2 Feature Extraction

This sub-section describes the methods to extract four types of features from the emotional speech database: pitch contour, speech timing, energy, and spectral features. These features have been individually used for vocal emotion recognition with certain success [1]. The audio processing is performed using Voicebox³, a speech processing toolbox from the Imperial College London [52]. It contains optimized MATLAB routines to perform speech analysis.

Before any feature is extracted, the audio files in the databases are pre-processed. Figure 4.1(a) shows an audio signal. The audio signal is divided into overlapping frames using Hamming windows with a 0.01 second time step. Then, the audio signal in each frame is processed using a voice activation detection (VAD) algorithm provided by Voicebox to detect the voiced parts of an audio signal. The algorithm calculates the probability that the audio signal is a human voice (Figure 4.1(b)) using the mean power of the frame's log-frequency spectrum and the sum of the power in the first three peaks.

The first feature type, pitch contour, is extracted using a pitch detection algorithm called Pitch Estimation Filter with Amplitude Compression (PEFAC) [53]. It involves the convolution of a pitch harmonic energy summing filter with the power spectrum in the log-frequency domain. This algorithm is used due to its robustness to a high level of noise. Figure 4.1(b) shows the pitch contour of the audio signal.

The second feature group is the short term energy E_s for a speech frame of a signal f with N_w data points that ends at signal point m :

$$E_s(m) = \frac{1}{N_w} \sum_{i=m-N_w+1}^m |f(i; m)|^2. \quad (4.3)$$

Figure 4.1(d) shows the energy contour plot of the audio signal.

---

³<http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html>

FIGURE 4.1: (a) An audio signal of a sample emotional speech; Features: (b) Voice probability (segments with probability more than 0.5 are considered voiced parts); (c) Pitch contour of the audio signal with blue portions indicating voiced parts, while red indicating otherwise; (d) Energy profile of the audio signal; (e) Speech pause time and speech duration; (f) Profiles for the first four Mel-Frequency Cepstral Coefficients (MFCCs).

Speech timing, which consists of speech rate and speech pause time (SPT), is the third feature group. The speech rate is the reciprocal of the duration of each voiced part in the speech signal, while speech pause time is the time gap between voiced parts of the signal (see Figure 4.1(e)).

Lastly, the spectral feature group contains Mel-Frequency Cepstral Coefficients (MFCCs) and is extracted from each frame of the audio signal using the Voicebox toolbox. MFCC measures the energy of certain frequency bands [1]. It is usually represented by twelve coefficients. They are the power cepstrum of an audio signal mapped to the Mel-scale. The power cepstrum of a signal is calculated using the equation below:

$$cepstrum = \left| \mathcal{F} \{ \log (\text{mel} (|\mathcal{F} \{ f(t) \} |^2)) \} \right|^2, \quad (4.4)$$

where $\mathcal{F}\{ \}$ represents Fourier transform and $\text{mel}(\ )$ is the Mel-scale mapping. Figure 4.1(f) shows the first four MFCCs of a speech signal.

A feature vector is formed from the statistics (mean and standard deviation) of the feature values above and their first and second difference, as shown in Table 4.1. The statistics are used to suit machine learning algorithms that do not consider temporal information [1]. The first and second differences are used to better reflect the temporal profile of the vocal features. In total, there are 40 features. The general variations of some of these features with basic emotions are listed in Table 4.2. It should be noted that not all the vocal features listed here will be exhibited by a speaker. Hence, feature selection is performed in a validation step discussed later.

TABLE 4.1: Detailed feature set (mean, variance, first and second difference) of each feature group

Feature Group	Features (#)	Statistics	#Features
Pitch	Pitch (1)	Mean & Variance	2
	Δ Pitch (1)	Mean & Variance	2
	Δ^2 Pitch (1)	Mean & Variance	2
Energy	Energy (1)	Mean & Variance	2
	Δ Energy (1)	Mean & Variance	2
	Δ^2 Energy (1)	Mean & Variance	2
Timing	Speech Rate (1)	Mean & Variance	2
	Δ Speech Pause Time (1)	Mean & Variance	2
Spectrum	MFCC (12)	Mean & Variance	24
Total			40

TABLE 4.2: General variation of vocal features with emotion [1, 2]. Symbols: (‘>’: increase; ‘<’: decrease; ‘=’: no change; ‘ μ ’: mean; ‘ σ ’: standard deviation; ‘ $E(\ )$ ’: energy; ‘ f_H ’: high frequency; ‘ f_M ’: medium frequency; ‘-’: not applicable)

Feature	Anger	Happiness	Sadness	Fear	Disgust
Pitch	$> \mu, > \sigma$	$> \mu, > \sigma$	$< \mu, < \sigma$	$>> \mu$	$< \mu$
Timing (Speech Rate)	$> \mu$	$> \mu$ or $< \mu$	$< \mu$	$> \mu$	$> \mu$
Intensity (Energy)	$> \mu$	$> \mu$	$< \mu$	$= \mu$	$< \mu$
Spectral (MFCC)	$> E(f_M)$	$> E(f_H)$	$< E(f_H)$	-	-

4.2.3.3 Dataset Division for Each Speaker

After obtaining suitable databases, sample speeches for each speaker are divided into three groups for the three standard procedures in machine learning: training, validation, and classification test. 60% are allocated randomly into the training set,

which is used to train the classifiers. 20% of the dataset is selected to be the validation set. It is used to optimize the classifiers in terms of features, machine learning algorithms, and parameter selection to avoid an over-fitting problem. Lastly, the remaining 20% is designated for the test set. It is used to measure the performance of the trained and optimized classifiers objectively. To avoid bias, it is important to ensure each group has a proportionate and sufficient number of samples for each emotion class.

4.2.3.4 Speaker Combination for SI, SD, and Hybrid Approaches

The selection of speakers for training and validation defines the SI, SD, and Hybrid approaches. In the SI approach, the training step uses training sets from a selected group of speakers (training speakers) while the validation step uses the validation sets from the same group of speakers. The unselected speakers become the test (target) speakers, which are used to test the performance of the trained classifier. For example (Table 4.3), in the case of the SAVEE database, we use the training set and validation set of Speaker 1 and 3 for training and validation, respectively. The test set of Speaker 2 is used as the test speaker for the vocal emotion classification test. A similar selection is performed in the EmoDB dataset. Training and validation use the training and validation sets from Speaker 1, 2, 3, 4, 5, and 6, while classification tests use test datasets from any of Speaker 7, 8, 9, and 10.

TABLE 4.3: Dataset division for SI approach. For fair comparison, only partial of the training data are used so that it matches the amount of data in other approaches. (Sp: Speaker; Tr: Training Set; CV: Cross-Validation Set; T: Test Set)

Database	Speaker-Independent (SI)		
	Training	Validation	Test
SAVEE	Sp 1,3 (Partial Tr)	Sp 1,3 (CV)	Sp 2 (T)
EmoDB	Sp 1,2,3,4,5,6 (Partial Tr)	Sp 1,2,3,4,5,6 (CV)	Sp 7/8/9/10 (T)

In the SD approach, only datasets from the same test speaker are used for training, validation, and classification tests. If the test speaker is Speaker 2, then the training set and validation set from Speaker 2 are used for training and validation, respectively (Table 4.4).

TABLE 4.4: Dataset division for SD approach. (Sp: Speaker; Tr: Training Set; CV: Cross-Validation Set; T: Test Set)

Database	Speaker-Dependent (SD)		
	Training	Validation	Test
SAVEE	Sp 2	Sp 2 (CV)	Sp 2 (T)
EmoDB	Sp 7	Sp 7 (CV)	Sp 7 (T)

The training and validation procedure for the Hybrid approach is slightly different from the standard procedure. It starts with an initial SI training, followed by validation, and ends with an extra training step: an SD training and validation. Both datasets from training speakers and test speakers are used separately for the two-stage training and validation. For example (Table 4.5), the training set and validation set of Speaker 1 and 3 (training speaker) of the SAVEE database are used for the initial SI training and validation. Then, the subsequent SD training uses the training set of Speaker 2.

TABLE 4.5: Dataset division for Hybrid approach. (Sp: Speaker; Tr: Training Set; CV: Cross-Validation Set; T: Test Set)

Database	Hybrid			
	Initial SI Training	Validation	Subsequent SD Training	Test
SAVEE	Sp 1,3 (11.1%, 22.2%, 33.3%, 44.4%, 55.56%, 66.6%, 77.8%, 88.9% Tr)	Sp 1,3 (CV)	Sp 2 (11.1%, 22.2%, 33.3%, 44.4%, 55.56%, 66.6%, 77.8%, 88.9% Tr)	Sp 2 (T)
EmoDB	Sp 1,2,3,4,5,6 (87.5%, 75%, 62.5%, 50%, 37.5%, 25%, 12.5% Tr)	Sp 1,2,3,4,5,6 (CV)	Sp 7/8/9/10 (12.5%, 25%, 37.5%, 50%, 62.5%, 75%, 87.5% Tr)	Sp 7/8/9/10 (T)

To ensure a fair comparison, the total number of training data sets for each approach must be about the same. Hence, while the SD approach can use all the samples from the training set of a speaker, the SI approach uses only a portion of the training sets from each training speaker, indicated by “Partial Tr” in Table 4.3. For the Hybrid approach, the training sets from both the training and test speaker

are further sub-sampled into 12.5%, 25%, 37.5%, 50%, 62.5%, 75%, and 87.5% of the original training dataset for EmoDB and 11.1%, 22.2%, 33.3%, 44.4%, 55.56%, 66.6%, 77.8%, and 88.9% for SAVEE to simulate the situation of limited data for SD training. The proportion of training data used between the initial SI training and subsequent SD training is such that the sum is 100%. For example, if 25% of the training sets of Speaker 1 and 3 are used, the corresponding proportion of the training set from Speaker 2 is 75%.

To increase confidence in the final result, we repeat the whole machine learning procedures (training, validation, and test) using different combinations of speakers. As there are ten speakers in EmoDB, if we select six speakers to be the training speakers and the rest as test speakers, there are 840 possible combinations of speakers. We chose 40 combinations for our experiment. Similarly, for SAVEE, we have three combinations from two training speakers and one test speaker.

4.2.3.5 Machine Learning Algorithm

Since the three approaches are algorithm independent, we chose Artificial Neural Networks (ANN) as they are one of the basic class of machine learning algorithms. Most importantly, they have been used in vocal emotion recognition [1, 54] with good results. Also, ANN is able to give non-linear classification with the use of hidden layers and non-linear activation functions. This non-linearity is crucial as some emotions like ‘Anger’ and ‘Happiness’ may not be linearly separable, given both are high arousal emotion. ANN has the drawback of higher training time. However, for our application, the computation time is still manageable as there are only 40 features. A regularization parameter is added to curb the over-fitting problem. This parameter is determined experimentally in the validation step.

4.2.3.6 Parameter Optimization and Feature Selection

Validation allows the optimization of the regularization parameter and vocal feature selection without over-fitting. Over-fitting is a problem where parameters are tuned so that the classification accuracy of training sets is high, but the performance cannot be generalized to other datasets (validation and test sets). To avoid this problem, for all three approaches and their speaker combinations, machine learning

is performed on training sets using 150 combinations of parameters and feature groups (listed in Table 4.6). Then, the trained classifiers are used to classify the validation sets, and the classification accuracy is used to determine the optimized parameters and features.

TABLE 4.6: Selection of vocal features and regularization parameter

Feature Group	Pitch, Speech Timing, Energy, MFCC
Parameter	0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10

4.2.3.7 Classification Test and Performance Comparison

For each approach in the final classification test, we used the optimized classifiers trained using different speaker combinations to classify emotion in the test sets. Then we calculated the average test accuracy over different speaker combinations for the three approaches.

4.2.4 Results

In this application, we aim to solve the dilemma between the low accuracy of the SI approach and the impracticality of the SD approach due to insufficient data by proposing a hybrid approach that merges the two. To prove the workability of the proposed approach, we have implemented the three approaches to classify emotion in two affective speech databases using vocal features. This sub-section shows the results of the implementation.

Figure 4.2 shows the performance comparison between the three approaches. The x-axis shows the amount of SD training data. 0% represents the SI approach, and 100% represents the SD approach. The Hybrid approach is situated between 0% and 100%. From the figure, we can observe that the SD approach has better classification accuracy (around 70%) than the SI approach (around 40%) for both databases. For the Hybrid approach, the accuracy lies in between, except at SAVEE 11.1% (35%). At around 45%, we observe that the accuracy has reached 65%, which is very close to SD accuracy (around 70%) for both databases. In other words, the hybrid approach uses a smaller number of SD training data to achieve an accuracy similar to SD. Lastly, it is noted that there are two kinks with lower accuracy at

FIGURE 4.2: Average test accuracy for different proportion of SD training data (SI: 0%; Hybrid: 0% < SD < 100%; SD: 100%)

both ends of the hybrid approach (transition from Hybrid to SI and SD). Further investigation is needed to understand this phenomenon.

4.3 Conclusion

In this chapter, a hybrid approach for vocal emotion recognition is proposed. The hybrid approach pre-trains a classifier using initial SI training data, followed by SD training. We implemented the SI, SD, and Hybrid approaches to compare their performance. The preliminary results show that the hybrid approach reduces the amount of SD training data required yet produces accuracy comparable to SD accuracy. The hybrid approach has the potential to make robotic vocal emotion recognition more practical as it needs less on-site training during deployment.

The hybrid approach is a crude form of transfer learning that initializes the parameters of a neural network using a larger generic dataset and performs subsequent retraining using a smaller dataset specific to the test condition. From the two tests using different vocal emotion recognition datasets, we showed that retraining with

a small amount of dataset from the test condition significantly pulls the classification performance closer to the classification results using the specific dataset alone.

In conclusion, the transfer learning method is effective at addressing variations in performance caused by subject difference. It is an in-learning method as it modifies the training process to adapt to the condition change, and it requires a certain amount of labeled data from the test condition.

Part II

Post-Learning Compensation

Chapter 5

Local Learning: Combining Nearby Predictions

In Chapter 4, we studied IWL and found that it is not only limited to batch learning, but the compensation can only be made batch-wise, making it inflexible to deal with a non-stationary condition that changes frequently. In this chapter, we explore the use of local learning to approximate IWL and make the compensation online in order to handle frequent shifts. Unlike IWL, local learning is a post-learning method that modifies the prediction process on the fly to compensate for the shift.

5.1 Motivation for Online Compensation

A timely response and robust prediction are essential in many machine learning applications. Ideally, we want a machine learning model to predict quickly and accurately, regardless of the change in the system dynamics being learned. These criteria are often contradictory, as exemplified in the two major ways to utilize machine learning algorithms. The first one performs one-off learning, and the learned model is deployed to make predictions. The second one incorporates machine learning algorithms in the deployed system and performs continuous learning to improve the prediction with time. In the first scenario, the learning is done offline while the prediction is online. Since the heavy learning workload is performed beforehand,

the online prediction phase is fast. However, as the learned model is fixed, it cannot adapt to non-stationary conditions. In the second scenario, both learning and prediction are online. Hence, the model can evolve with changes in conditions, but at the expense of increased online computation time.

Under the above framework, importance weighted learning (IWL) [13] can be categorized into the first type. However, unlike other methods, it is designed to adapt its prediction to change in input distribution (covariate shifts) [9]. In doing so, it forgoes its ability to perform online prediction as it uses test data input in the learning process to estimate the importance weights and assumes test data are available in batch. As a result, IWL is not suitable for applications like robot perception, where data come in streams and require immediate response [12].

One may suggest accumulating the streaming data and process the test data later, but in many applications, especially in human-robot interaction (HRI), a timely response is required. This calls for instant processing of the incoming single test data point (online prediction). Consider the following human-robot interaction scenario: a robot is in a conversation with a human being. The speech recognition software in the robot should process an incoming speech immediately so that a timely response can be formulated after the human speaker finishes his or her sentence. Waiting for a few more speeches before processing causes a delay in the response and makes the human-robot interaction awkward [55].

Besides, models learned under IWL algorithms are only tuned to a specific covariate shift exhibited by the test data used in training. The model cannot handle new data with a different covariate shift. Under this situation, re-training using the new test data is required. Furthermore, in practical situations, we should expect the shift to change with time, similar to concept shifts [56]. This varying shift requires constant re-training, which is computationally expensive.

Hence, existing IWL algorithms are not practical in some real-world applications which require online prediction under covariate-shift conditions. As a result, we need to develop an online algorithm to compensate for covariate shifts. This chapter proposes the use of local learning algorithms to address online prediction under covariate shifts, for it can be shown that local learning approximates IWL in an online setting.

5.2 Concept

5.2.1 Local Learning

Local learning works by confining learning to small regions of input space. It helps to achieve a better bias-variance trade-off by adapting parameters according to local regions [57]. It is also used to limit the generalization capability of a local model to avoid negative interference [58]. We observe that there is a similarity in the physical interpretation of local learning and IWL. Both work on the premise that training data similar to test data points are more relevant in predicting on the test data [21, 58]. This observation provides a hint that local learning can be used for covariate-shift problems. In fact, the weighted empirical risk minimization (ERM) formulation of locally-weighted learning (LWL) is almost the same as IWL (see Equation 1.11), and the only difference is in the weight, W_L :

$$\min_{\hat{f} \in \mathcal{F}} \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} l\left(\hat{f}(\mathbf{x}_i) - y_i\right) W_L(\mathbf{x}_i, \mathbf{x}). \quad (5.1)$$

The weight is a kernel function that weights the training data based on the distance of training data from a query point:

$$W_L(\mathbf{x}_i, \mathbf{x}) = K_h(\|\mathbf{x}_i - \mathbf{x}\|), \quad (5.2)$$

where the query point, $\mathbf{x}$, is a test feature for which we want to predict the output; $\|\mathbf{x}_i - \mathbf{x}\|$ is the distance function between a training data point, $\mathbf{x}_i$, and the query point, $\mathbf{x}$; and h is the bandwidth of a kernel function, K_h . If a square loss:

$$l\left(\hat{f}(\mathbf{x}_i) - y_i\right) = \left(\hat{f}(\mathbf{x}_i) - y_i\right)^2 \quad (5.3)$$

is used in Equation 5.2, the prediction is given by the weighted combination of the training labels, y_i :

$$\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{n_{tr}} y_i W_L(\mathbf{x}_i, \mathbf{x})}{\sum_{i=1}^{n_{tr}} W_L(\mathbf{x}_i, \mathbf{x})}. \quad (5.4)$$

In this case, weighting the criterion is equivalent to weighting the training labels [58]. By using this weighting, LWL forms a local model around a query point

to predict the output of the query point. In this way, the learning and prediction are all achieved online for each query point.

In LWL, optimization of the kernel bandwidth is performed for each new incoming test datum to better adapt to changes in training-data density, model capacity, and noise level. This could be computationally expensive. For regression problems, the leave-one-out-cross-validation (LOOCV) optimization scheme can be solved analytically, and this speeds up the optimization process. For classification problems, there is no similar counterpart [59].

In addition, LWL is an instance-based method, where the prediction for a query point is just a weighted combination of the labels from the training data. There is no model abstraction from the data. As the training data increases over time, the amount of computation for prediction increases as well. Similarly, more computation is needed for the optimization process to determine the kernel width.

To avoid heavy online computation due to the increasing number of training data points, another type of local learning algorithm which constructs a group of local models [60] can be employed. It partitions the input space into a few local neighborhoods and learns a local model for each neighborhood. Due to this nature, this class of algorithms is called multi-local-model learning (MLML) in this thesis. In contrast to LWL, the local models in MLML are constructed to represent the data in the neighborhoods, instead of using the data labels directly. With this modification, the prediction in Equation 5.4 changes to a weighted combination of predictions from n local models, $\tilde{f}_k(\mathbf{x})$:

$$\hat{f}(\mathbf{x}) = \frac{\sum_{k=1}^n \tilde{f}_k(\mathbf{x}_i) W_L(\mathbf{x}_k^c, \mathbf{x})}{\sum_{k=1}^n W_L(\mathbf{x}_k^c, \mathbf{x})}, \quad (5.5)$$

where $\mathbf{x}_k^c$ is the centroid of k -th local model. This change avoids the data accumulation issue. Also, the method does not need to optimize the kernel width over all the training data. All these qualities make the online prediction much faster. This class of algorithms includes radial basis function (RBF) networks with locally linear models [61], self-organizing map (SOM) with locally linear models [62], and mixture-of-expert (MoE) models [63].

5.2.2 Making IWL Online

As discussed in Chapter 3, the estimation of the density ratio requires both training and test data in batches. These methods fail when test data arrive sequentially, and each datum requires instant processing (online learning). This failure is because they cannot estimate the density ratio with only a single test data point. In this sub-section, we develop an online IW estimation algorithm to estimate the density ratio on the fly, with only a single data point.

To achieve online IW estimation, we first use the kernel density estimation (KDE) method to estimate the IW in the weighted ERM formulation similar to Equation 5.1:

$$\min_{\hat{f} \in \mathcal{F}} \left[\frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \frac{\frac{1}{n_{te}} \sum_{j=1}^{n_{te}} K_h^{te}(\|\mathbf{x}_i - \mathbf{x}_j\|)}{\frac{1}{n_{tr}} \sum_{k=1}^{n_{tr}} K_h^{tr}(\|\mathbf{x}_i - \mathbf{x}_k\|)} l(\hat{f}(\mathbf{x}_i) - y_i) \right], \quad (5.6)$$

where $l(\hat{f}(\mathbf{x}_i) - y_i)$ is the loss function evaluated at data point i , n_{tr} and n_{te} are the numbers of training and test data, and K_h is a kernel function. The equation shows that it is possible to decompose the expected generalization error into individual terms according to individual test data. Instead of minimizing the sum of all test data term, we minimize each individual term:

$$\min_{\hat{f} \in \mathcal{F}} \left[\frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} \frac{K_h^{te}(\|\mathbf{x}_i - \mathbf{x}_j\|)}{\frac{1}{n_{tr}} \sum_{k=1}^{n_{tr}} K_h^{tr}(\|\mathbf{x}_i - \mathbf{x}_k\|)} l(\hat{f}(\mathbf{x}_i) - y_i) \right], \text{ for } j = 1, \dots, n_{te}. \quad (5.7)$$

In Equation 5.7, the test-data density for a single test data point is assumed to be a kernel function. In this way, we can perform importance weighting for a single point. The process becomes online! It can also improve the prediction accuracy under a covariate-shift condition because minimizing the individual terms could further reduce the overall bias. However, according to the bias-variance trade-off principle, the variance could increase as well. Hence, proper regularization and model selection should be put in place to select kernel bandwidth, h , for each test data point.

Another advantage of making IWL online is that it allows the method to deal with time-varying covariate shifts. Existing batch IWL methods assume the availability of all the shifted test data beforehand, and hence the test-data density is fixed while calculating the one-time importance weight. The resulting importance weighted model is subsequently adapted to only this particular shift. In contrast to batch

methods, the online formulation does not assume the availability of all the test data beforehand. It recalculates the weights for each incoming test data. In this way, the weights always reflect the change in the test-data density and adapt the model to the shift accordingly.

Based on the formulation in Equation 5.7, we designed a naive meta-algorithm to make use of existing importance estimation methods for online IW estimation. It works by generating artificial test data points surrounding a test data point according to a kernel density function. Together with the training data, these artificial test data are then used by one of the existing IW estimation algorithms, such as KMM, KLIEP, and uLSIF discussed in Chapter 3 to estimate the importance weights. This allows simple conversion from current batch IWL methods to online versions. Similar to IWL, it also allows the flexibility to use any ERM-based learning method. Since IWL works under the assumption of a mis-specified model, a linear model is sufficient. A simpler model has the advantage of better interpretability and extension to high dimensional problems. The meta-algorithm is shown in Algorithm 3.

Algorithm 3: Naive Online IWL (OIWL)

Input: training data, $\{\mathbf{x}_{tr}^{(i)}, y_{tr}^{(i)}\}_{i=1}^{n_{tr}}$; test features, $\{\mathbf{x}_{te}^{(j)}\}_{j=1}^{n_{te}}$.

Hyperparameters: number of artificial data, m ; kernel width, h .

Output: predicted test labels, $\{\hat{y}_{te}^{(j)}\}_{j=1}^{n_{te}}$.

for $j = 1, \dots, n_{te}$, **do**

    Set $\hat{p}_{te}(\mathbf{x}) = K_h^{te}(\|\mathbf{x} - \mathbf{x}_{te}\|)$.

    Generate m data points, $\{\hat{\mathbf{x}}_{te}^{(k)}\}_{k=1}^m \sim \hat{p}_{te}$.

    Estimate $W_I(\mathbf{x})$ using $\{\mathbf{x}_{tr}^{(i)}\}_{i=1}^{n_{tr}}$ and $\{\hat{\mathbf{x}}_{te}^{(k)}\}_{k=1}^m$.

    Learn $\hat{f}(\mathbf{x}) = \arg \min_{f \in \mathcal{F}} \left[\frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} l\left(\hat{f}(\mathbf{x}_{tr}^{(i)}) - y_{tr}^{(i)}\right) W_I(\mathbf{x}_{tr}^{(i)}) \right]$, for

$i = 1, \dots, n_{tr}$.

    Predict $\hat{y}_{te}^{(j)} = \hat{f}(\mathbf{x}_{te}^{(j)})$.

end for

In the algorithm, we need to set the bandwidth of the density kernel, h , for each test data point. As mentioned before, the test-data density support must be within the training-data density support. Therefore, for kernel bandwidth selection, the underlying intuition is to make the density function covers sufficient nearby training instances so that we can estimate the density ratio without the issue of inverting an almost-singular matrix. To achieve this, the kernel bandwidth is set to be the

mean distance of k nearest neighbors. As the choice of kernel function does not matter in practice, Gaussian kernels are used.

Another parameter is the number of artificial test data, m . We should set m to a large number so that the IW estimation is consistent. However, this will increase the computational load. A compromise between the two considerations has to be made. In our case, we set it to a number slightly larger than the number of training data.

Since we only need to classify one test data point at any time, a simple classifier class, $\mathcal{F}$, such as a linear classifier, is sufficient. This decreases the computation time for learning. However, the savings in computation time by the use of this simpler classifier may be offset by the need to perform weighted training for each incoming test data point. In each classification, the algorithm calculates the importance weights using all the training data even though some data are irrelevant in classifying the test data point. This step is computationally expensive. This issue leads us to the next sub-section, where we attempt to approximate IWL by local learning.

5.2.3 Approximating Online IWL using Local Learning

As discussed in Sub-Section 5.2.1, local learning is similar to IWL because both use weighting to implicitly select training data that are similar to test data points. Given the similarity, we suggest that local learning can be used for covariate shift problems. Moreover, a similar local weighting concept has been used in lazy learning to deal with time-varying input sampling shifts [64] (a generalized covariate shift [65]). In addition, [33] mentions that KMM is related to local learning. Hence, in this sub-section, we examine the connection between local learning and OIWL. We then propose the use of local learning algorithms in covariate-shift problems based on this connection.

In OIWL, we approximate the test-data density of a single test data point, $\mathbf{x}_{te}$, as a narrow symmetric kernel function centered at the test data point and with a kernel bandwidth, h :

$$p(\mathbf{x})_{te} \approx K_h(\|\mathbf{x} - \mathbf{x}_{te}\|). \quad (5.8)$$

If we keep h small relative to the support of the training data distribution, the density value of the test data will be much higher than the training-data density:

$$p(\mathbf{x})_{te} \gg p(\mathbf{x})_{tr}. \quad (5.9)$$

Within the narrow test-data density kernel, the variation of training-data density values along the radial direction of the kernel is small compared to the change of test-data density values along the same direction. As a result, the importance-weight function (ratio of test-data density to training-data density) is more sensitive to the variation of test-data density, because the values of training-data density within the kernel are relatively constant. By treating the training-data density function as a constant ($1/p(\mathbf{x})_{tr} \approx C$), we can approximate the importance weight function by a scaled distance-based kernel function centered at the test data point:

$$W_I = \frac{p(\mathbf{x})_{te}}{p(\mathbf{x})_{tr}} \approx Cp(\mathbf{x})_{te} \approx CK_h(\|\mathbf{x} - \mathbf{x}_{te}\|). \quad (5.10)$$

Hence, in the online situation, we can approximate IWL using LWL. Following this argument, LWL should work well, too, in covariate-shift situations.

To implement LWL, we can modify Algorithm 3 by replacing W_I with a distance-based kernel weight, W_L . Similar to OIWL, we use the average of k nearest neighbors to determine the kernel bandwidth in LWL. This step is to adapt to the variation of training data density in the input space. In IWL, the training-data density is a factor in the calculation of importance weights, whereas, for LWL, the local weight only considers the distance from the test data point. The use of a few nearest neighbors serves as an indicator of the training-data density. Compared to IWL, the calculation of weights is simpler in LWL.

Besides sharing the common interpretation of their weighting scheme, OIWL and LWL share the same computational expense problem. First, both algorithms perform weighted learning using all the training data again and again for each incoming test data point. In addition, cross-validation is usually performed to obtain optimal regularization parameters. These processes are computationally expensive and affect the ability of the algorithms to provide a timely online prediction. Hence, we want to decouple the learning and prediction, as we shall see in the next sub-section.

5.2.4 Multi-Local-Model Learning (MLML)

Since we have established the link between OIWL and local learning through LWL, we can further explore the use of the second type of local learning algorithms to address covariate-shift problems. This type of local learning un-bundles the learning and prediction stage, while not affecting the ability to adapt to changing shift. It is also less computationally expensive for online computation.

MLML has a separate learning and prediction stage. In the learning stage, it partitions the feature space into multiple sections and trains a local model for each section. Next, for each test data point, it combines predictions from the local models to give a final prediction. Note that these two stages can be done independently, unlike LWL, where each prediction has to be preceded by a learning process. With learning and prediction decoupled, we have the flexibility to perform batch learning or online learning. This means we can pre-train the prediction model before making any predictions if there are training data available beforehand. The learning can be continued whenever new training data are available.

By creating multiple local models, MLML algorithms do not suffer increasing online computation time as the training data accumulate. MLML achieves faster online computation by representing the feature space with multiple local models instead of using all the accumulated training data. In the learning stage, the local models are trained using only part of the training data that are relevant to the models. This logic applies similarly to hyperparameter tuning. Similarly, only a few local models are triggered to provide predictions. In a nutshell, it suffices to say that at any stage, be it learning or prediction, only a few local models are involved, instead of the full training dataset. This partition makes the online computation time stay constant over time.

The local models also allow predictions to be adaptive to covariate shifts. In the prediction phase, only nearby models are used for prediction. Instead of weighting all the training data points, it weights the predictions from the nearby local models. Hence, the method retains the adaptive characteristic of local learning in compensating for covariate shifts.

As reviewed in Section 5.2.1, there are a few choices for MLML algorithms. We chose to use SOM-based algorithms in this chapter because they are able to approximate a feature space with a map in a lower dimension that still reflects the topological ordering and density of the original input data space [66].

As illustrated in Algorithm 4, the SOM-based MLML algorithms are divided into three parts. The first part performs SOM clustering on training data. In the second part, it trains a local model for each SOM cluster. With the trained local models, we perform online prediction in the third part. The prediction is the weighted average of the predictions of all the local models. The weight is calculated based on the concept of receptive fields [60] and measures the activation of a local model on a test data point. To achieve this, a distance-based kernel function is assigned to an SOM node. The center of the kernel is at the data mean of the SOM node, while the kernel width is set to be the variance of the data in the node, which reflects the width of the SOM cluster.

Algorithm 4: SOM-MLML

Input: training data, $\{\mathbf{x}_{tr}^{(i)}, y_{tr}^{(i)}\}_{i=1}^{n_{tr}}$; test features, $\{\mathbf{x}_{te}^{(j)}\}_{j=1}^{n_{te}}$.

Hyperparameters: average number of samples in a node, n_{avg} .

Output: predicted test labels, $\{\hat{y}_{te}^{(j)}\}_{j=1}^{n_{te}}$.

SOM Clustering:

Set $n = \left\lceil \sqrt{n_{tr}/n_{avg}} \right\rceil$.

Construct $n \times n$ SOM, $\{\delta_k, k = 1, \dots, n^2\} = \text{SOM} \left(\{\mathbf{x}_{tr}^{(i)}, y_{tr}^{(i)}\}_{i=1}^{n_{tr}} \right)$.

Calculate mean $\boldsymbol{\mu}_k$ and variance $\boldsymbol{\sigma}_k^2$ for $\delta_k, k = 1, \dots, n^2$.

Supervised Learning:

for $k = 1, \dots, n^2$, **do**

Learn local model, $\hat{f}_k$ from $\{\mathbf{x}, y\} \in \delta_k$.

end for

Online Prediction:

for $j = 1, \dots, n_{te}$, **do**

Obtain $\tilde{y}_{jk} = \hat{f}_k \left(\mathbf{x}_{te}^{(j)} \right), k = 1, \dots, n^2$.

Set $W_L^{(jk)} = K_h \left(\left\| \mathbf{x}_{te}^{(j)} - \boldsymbol{\mu}_i \right\| \right)$.

Predict $\hat{y}_{te}^{(j)} = \frac{\sum_{k=1}^{n^2} W_L^{(jk)} \tilde{y}_{jk}}{\sum_{k=1}^{n^2} W_L^{(jk)}}.$

end for

5.3 Application 1: Inverse Kinematic Modeling

To compare the performance between IWL and local learning in terms of regression, we revisit the inverse kinematics problem from Chapter 3.

5.3.1 Method

In this application, the local learning algorithm used is the locally-weighted least-squared (LWKLS) regression, as shown in Algorithm 5. Gaussian kernels are used in this algorithm, and the kernel bandwidth, h , is determined through LOOCV for each query point.

One feature worth noting in Algorithm 5 is that for each incoming test data point, both learning and prediction are performed. This feature has the advantage of being able to handle changing covariate shifts. Unlike IWL in Chapter 3, where predictions for each red circle is made in batches and separately, the prediction of LWKLS can be made online on any red circle and in any order. However, performing learning for each test data point is computationally expensive, especially when the training dataset is large.

Algorithm 5: Locally-Weighted Least-Squared Regression (LWKLS)

Input: training data, $\{\mathbf{x}_{tr}^{(i)}, y_{tr}^{(i)}\}_{i=1}^{n_{tr}}$; test feature, $\{\mathbf{x}_{te}^{(j)}\}_{j=1}^{n_{te}}$.

Output: predicted test labels, $\{\hat{y}_{te}^{(j)}\}_{j=1}^{n_{te}}$.

Algorithm:

for $j = 1, \dots, n_{te}$, **do**

Form local weight matrix: $W_{i,i} = \exp\left(-\frac{\|\mathbf{x}_{tr}^{(i)} - \mathbf{x}_{te}^{(j)}\|^2}{2h^2}\right)$.

Form regression model parameter vector: $\hat{\beta} = (\mathbf{X}_{tr}^T \mathbf{W} \mathbf{X}_{tr})^{-1} \mathbf{X}_{tr}^T \mathbf{W} \mathbf{Y}_{tr}$.

Predict test labels: $\hat{y}_{te}^{(j)} = \hat{\beta}^T \mathbf{x}_{te}^{(j)}$.

end for

5.3.2 Results and Discussion

We append the prediction results by LWKLS to the original results from Figure 3.4 to compare the regression performance of the unweighted, importance-weighted, and locally-weighted kernel least-squared algorithms (see Figure 5.1).

Using the MSE contour of 0.001 as a baseline, we can see the areas bounded by the 0.001-contour in AIWKLS and LWKLS increases. In both cases, the prediction accuracy improved even in areas with fewer training data (covariate shifts). From this observation, we can conclude that both AIWKLS and LWKLS help to compensate for the shifts. However, for LWKLS, there is a rapid drop in accuracy in the upper right corner. This failure is due to a near-singular matrix inversion in LWKLS. It is caused by very small local weights when the query point is far away from the training data points (see Algorithm 5). An instance of prediction with this ill-condition is shown in Figure 5.2.

From Figure 5.1, we observe that the variance of the regression method increases from KLS to LWKLS. The increase in variance is a natural consequence of the increase in model capacity.

5.4 Application 2: Vocal Emotion Recognition

We applied the local learning technique to the same vocal emotion recognition problem described in Chapter 4.

5.4.1 Label Modification

To simplify the vocal emotion classification problem for easier analysis, we reduce the number of labels from seven to four. They are anger, sadness, happiness, and neutral. In addition to the basic-emotion classification in Chapter 4, two additional emotion classification types, arousal and valence, are introduced. While the six-basic-emotion model treats each emotion as a discrete category, the alternative model defines emotion according to dimensions. Arousal and valence are the two most used dimensions. Arousal measures how energizing (low or high) an emotion is, while valence gauges an emotion on a scale of positive and negative.

FIGURE 5.1: Results of inverse kinematics prediction with and without covariate shift compensation

FIGURE 5.2: Erroneous prediction by LWKLS when the covariate shift is large: (a) Large covariate shift: Test task space (red) is further away from the training task space (blue); (b) AIWKLS: Inaccurate but consistent; (c) LWKLS: Inaccurate and inconsistent

For the additional two classifications, we regrouped the six basic emotions along these two dimensions. For simplicity, we employed binary classification. Arousing emotions such as anger, happiness, fear, and surprise are classified as high arousal emotions, while the rest, like disgust, sadness, and neutral, are grouped as low arousal emotions. Emotions with positive valence include happiness and neutral while the negative ones are anger, disgust, fear, and sadness. Note that the valence of surprise can be both positive and negative. Hence, the surprise emotion is excluded in valence classification.

5.4.2 Detecting Covariate Shift

We use the failure rate of the two-sample Kolmogorov-Smirnov (KS) Test [67] to indicate if a covariate shift occurs. The KS-Test tests if two one-dimensional datasets are from the same continuous distribution. To test for covariate shifts in multidimensional data, the test is performed on each dimension, and the proportion of dimensions that fail the test over the total number of dimensions are calculated as the failure rate. A large KS-Test failure rate indicates that the difference in training and test feature space is large. This indicates a potential covariate shift. If the accuracy is affected by the shift (below average), there is a chance we can further improve the result by applying covariate-shift compensation methods such as IWL and local learning methods.

5.4.3 Classification Methods

We tested the three classifications using unweighted, importance-weighted, and locally weighted methods. The base classification algorithm used is logistic regression (LR). Importance weights are generated using unconstrained least squared importance fitting (uLSIF) [13] (see Algorithm 1) and incorporated into the base algorithm to produce importance-weighted logistic regression (IWLR). For local learning, an SOM-MLML algorithm (see Algorithm 4) was used to combine a self-organizing map and logistic regression (SOMLR).

The classifications were carried out on both SAVEE and EmoDB datasets. Correlation-based feature selection was employed for all three classification to select relevant features for the classification. A leave-one-subject-out scheme was used to create

different training and test conditions. Under this scheme, data from one subject are held out as the test data, while the data from other subjects are used for training. Lastly, we performed a KS Test between the training and test data to measure the difference in the distribution of the feature space.

5.4.4 Results and Discussion

Tables 5.1, 5.2, and 5.3 show the arousal, valence, and four-basic-emotion classification results of the EmoDB dataset, respectively. The KS-Test failure rates are shown in the tables, too. From the tables, we see that for the arousal classification, Subjects 10, 11, 12, and 16 have a higher-than-average KS-Test failure rate and below-average classification accuracy. Note that though Subject 8 has a higher-than-average KS-Test failure rate, the classification accuracy does not seem to suffer as it is above the average. Similarly, Subject 9 has a below-average classification accuracy but below-average KS-Test failure rate, indicating that other factors besides covariate shifts could be at play.

TABLE 5.1: Arousal emotion classification of the EmoDB dataset using logistic regression (LR) and the KS-Test failure rate: Subject 10, 11, 12, and 16 (in bold) have higher-than-average KS-Test failure rates and below-average classification accuracy.

Arousal		
Subject	KS Test	Classification (%)
		LR
3	0.29	100.0
8	0.39	96.6
9	0.22	93.0
10	0.39	92.1
11	0.41	87.3
12	0.37	88.6
13	0.36	95.1
14	0.35	98.6
15	0.35	100.0
16	0.50	94.4
Average	0.36	94.6

The KS-Test failure rates are shown in the tables, too. From the tables, we see that for the arousal classification, Subjects 10, 11, 12, and 16 have a higher-than-average KS-Test failure rate and below-average classification accuracy. Note that

TABLE 5.2: Valence emotion classification of the EmoDB dataset using logistic regression (LR) and the KS-Test failure rate: Subject 12, 13, 14, 15, and 16 (in bold) have higher-than-average KS-Test failure rates and below-average classification accuracy.

Valence		
Subject	KS Test	Classification (%)
		LR
3	0.32	81.6
8	0.34	87.9
9	0.17	79.1
10	0.39	84.2
11	0.34	76.4
12	0.43	74.3
13	0.46	78.7
14	0.41	81.2
15	0.43	73.2
16	0.68	77.5
Average	0.40	79.4

TABLE 5.3: Four-basic-emotion classification of the EmoDB dataset using logistic regression (LR) and the KS-Test failure rate: Subject 3, 11, and 16 (in bold) have higher-than-average KS-Test failure rates and below-average classification accuracy.

Four Basic Emotion		
Subject	KS Test	Classification (%)
		LR
3	0.41	84.6
8	0.34	85.7
9	0.26	86.7
10	0.18	90.5
11	0.39	82.9
12	0.31	100.0
13	0.34	86.1
14	0.35	82.9
15	0.29	91.2
16	0.65	74.4
Average	0.35	86.5

though Subject 8 has a higher-than-average KS-Test failure rate, the classification accuracy does not seem to suffer as it is above the average. Similarly, Subject 9 has a below-average classification accuracy but below-average KS-Test failure rate, indicating that other factors besides covariate shifts could be at play.

For the valence classification, the subjects potentially affected by covariate shifts

are 12, 13, 14, 15, and 16. Out of these subjects, Subject 16 experienced the highest shift, as 68% of the feature dimensions failed the KS test. Subject 14 experienced a covariate shift (higher-than-average KS-Test failure rate), but the classification performance is unaffected (above-average accuracy). On the other hand, Subject 9 experiences little covariate shifts as the failure rate is only 0.17 compared to the average value of 0.40, and its classification performance at 79.1% is only slightly worse than the average at 79.4%. Covariate-shift compensation methods may not apply to these two scenarios.

Lastly, for the four-basic-emotion classification, the subjects fulfilling the two criteria are Subject 3, 11, and 16. Subject 13 has a slightly smaller accuracy than the average (86.1% compared to 86.5%), and its KS-Test failure rate is slightly better than the average (0.34 compared to 0.35). Also worth noting is Subject 16, which has more than half of its feature dimensions failing the KS-Test, indicating a large covariate shift. The corresponding classification result shows that it has the lowest accuracy, probably due to the large covariate shift.

Comparing the three classifications, we see little overlap in subjects that fulfill the two covariate-shift criteria except Subject 16. The difference could be explained by the use of a correlation-based feature selection procedure. Due to this extra step, the three classifications use different feature combinations.

With the assumption that the subjects who fulfill the two criteria potentially suffer from covariate shifts, covariate-shift compensation techniques were applied in these cases. Table 5.4, 5.5, and 5.6 show the classification results of affected subjects in the EmoDB dataset using both the importance-weighted and SOM-based version of logistic regression.

Checking through the classification results of those subjects suspected to be affected by covariate shifts, we can observe that the local learning method often improves the classification performance. The only exception is Subject 16 in the four-basic-emotion classification, where the performance drops. This could be due to the excessive covariate shift indicated by the very large KS failure rate at 0.65, compared to the average at 0.35.

While the compensation results by SOMLR are convincing, the performance of IWLR is less so. In most of the cases, the performance is the same as unweighted

TABLE 5.4: Arousal classification of EmoDB subjects that suffered from covariate shifts using logistic regression (LR), importance-weighted logistic regression (IWLR), and SOM-based logistic regression (SOMLR): Methods with the highest accuracy for each subject are highlighted in bold. SOMLR improves the performance of all affected subjects but IWLR does not.

Arousal			
Subject	Classification (%)		
	LR	IWLR	SOMLR
10	92.1	92.1	94.7
11	87.3	80.0	89.1
12	88.6	88.6	94.3
16	94.4	94.4	94.6
Average	90.6	88.8	93.2

TABLE 5.5: Valence classification of EmoDB subjects that suffered from covariate shifts using logistic regression (LR), importance-weighted logistic regression (IWLR), and SOM-based logistic regression (SOMLR): Methods with the highest accuracy for each subject are highlighted in bold. Both SOMLR and IWLR improve or maintain classification performance but SOMLR performs the best.

Valence			
Subject	Classification (%)		
	LR	IWLR	SOMLR
12	74.3	80.0	80.0
13	78.7	78.7	78.7
15	73.2	75.0	75.0
16	77.5	77.5	78.9
Average	75.9	77.8	78.1

classification. In two instances (Subject 11 in the arousal classification and Subject 16 in the four-basic-emotion classification), the performance became worse.

We repeated the same experiment but with the SAVEE dataset. Table 5.7 shows the unweighted classification and KS-Test results of all three types of vocal emotion classifications. Unlike the previous dataset, the subjects that fulfill the covariate-shift criteria are the same throughout the three classification. Also, the KS-Test failure rates are relatively high compared to the EmoDB dataset, which explains the relatively low accuracy.

From Table 5.8, we see that SOMLR further improves the classification results of subjects potentially affected by covariate shifts, except for Subject 4 in the four-basic-emotion classification. Again, we observe that this drop in performance happened when the KS-Test failure rate is large (0.88). Unlike the EmoDB dataset,

TABLE 5.6: Four-basic-emotion classification of EmoDB subjects that suffered from covariate shifts using logistic regression (LR), importance-weighted logistic regression (IWLR), and SOM-based logistic regression (SOMLR): Methods with the highest accuracy for each subject are highlighted in bold. SOMLR generally improves the performance except for Subject 16. IWLR only improves the performance of Subject 3.

4 Basic Emotion			
Subject	Classification (%)		
	LR	IWLR	SOMLR
3	84.6	87.2	87.2
11	82.9	82.9	88.6
16	74.4	66.7	64.1
Average	80.6	78.9	80.0

TABLE 5.7: Arousal, valence, and four-basic-emotion classification of the SAVEE dataset using logistic regression and the KS-Test failure rate: Subject 3 and 4 (in bold) have higher-than-average KS-Test failure rates and below-average classification accuracy.

Subject	Arousal		Valence		4 Basic Emotion	
	KS Test	Classification	KS Test	Classification	KS Test	Classification
		LR (%)		LR (%)		LR (%)
1	0.73	95.8	0.60	78.1	0.58	77.3
2	0.74	87.5	0.76	81.0	0.71	82.7
3	0.84	67.5	0.74	57.1	0.73	32.0
4	0.85	50.0	0.84	44.8	0.88	42.7
Average	0.79	75.2	0.73	65.3	0.72	58.7

we also observe that the importance-weighted learning worked well in this dataset. In fact, it not only improves the classification accuracy but achieves the best results.

TABLE 5.8: Arousal, valence, and four-basic-emotion classification of SAVEE subjects that suffered from covariate shifts using logistic regression, importance-weighted logistic regression, and SOM-based logistic regression: Methods with the highest accuracy for each subject are highlighted in bold. On average, both SOMLR and IWLR improve the classification results. IWLR performs the best and SOMLR’s performance reduces for Subject 4 in the four-basic-emotion classification.

Subject	Classification (%)								
	Arousal			Valence			Four Basic Emotion		
	LR	IWLR	SOMLR	LR	IWLR	SOMLR	LR	IWLR	SOMLR
3	67.5	81.7	68.3	57.1	60.0	67.6	32.0	56.0	57.3
4	50.0	50.0	51.7	44.8	61.9	52.4	42.7	45.3	33.3
Average	58.8	65.8	60.0	51.0	61.0	60.0	37.4	50.7	45.3

The results shown in Table 5.4, 5.5, 5.6, and 5.8 prove that when covariate shifts occur, both importance-weighted and local learning approaches are effective to compensate for the drop in performance. The next question is, how about those cases not affected by covariate shifts?

Tables 5.9, 5.10, 5.11, and 5.12 show the classification results for subjects in the EmoDB and SAVEE datasets that do not fulfill the two covariate shift criteria. The results are mixed. There is no single method that conclusively outperforms the others. In some instances, the weighted logistic regression performs better than the unweighted one, such as for the arousal classification in both EmoDB and SAVEE datasets; and valence classification in the EmoDB dataset. On average, the unweighted method performs the best, especially in the valence classification in the SAVEE dataset; and four-basic-emotion classification in both EmoDB and SAVEE datasets. It shows that weighting-based compensation methods for covariate shifts are restrictive.

TABLE 5.9: Arousal classification of EmoDB subjects that did not suffer from covariate shifts using logistic regression, importance-weighted logistic regression, and SOM-based logistic regression: Methods with the highest accuracy for each subject are highlighted in bold. IWLR has the highest mean accuracy.

Arousal			
Subject	Classification (%)		
	LR	IWLR	SOMLR
3	100.0	100.0	100.0
8	96.6	98.3	94.8
9	93.0	93.0	95.3
13	95.1	95.1	93.4
14	98.6	98.6	98.6
15	100.0	100.0	90.7
Average	97.2	97.5	95.5

TABLE 5.10: Valence classification of EmoDB subjects that did not suffer from covariate shifts using logistic regression, importance-weighted logistic regression, and SOM-based logistic regression: Methods with the highest accuracy for each subject are highlighted in bold. SOMLR has the best classification performance on average.

Valence			
Subject	Classification (%)		
	LR	IWLR	SOMLR
3.0	81.6	77.6	73.5
8.0	87.9	86.2	82.8
9.0	79.1	79.1	90.7
10.0	84.2	84.2	89.5
11.0	76.4	80.0	76.4
Average	81.8	81.4	82.6

TABLE 5.11: Four-basic-emotion classification of EmoDB subjects that did not suffer from covariate shifts using logistic regression, importance-weighted logistic regression, and SOM-based logistic regression: Methods with the highest accuracy for each subject are highlighted in bold. LR has the best classification performance.

4 Basic Emotion			
Subject	Classification (%)		
	LR	IWLR	SOMLR
8	85.7	85.7	83.3
9	86.7	86.7	83.3
10	90.5	85.7	90.5
12	100.0	100.0	81.8
13	86.1	83.3	94.4
14	82.9	78.0	70.7
15	91.2	88.2	79.4
Average	89.0	86.8	83.4

TABLE 5.12: Arousal, valence, and four-basic-emotion classification of SAVEE subjects that suffered from covariate shifts using logistic regression, importance-weighted logistic regression, and SOM-based logistic regression: Methods with the highest accuracy for each subject are highlighted in bold. On average, both SOMLR and IWLR improve the classification results. There is no consistent winner that performs the best in all cases.

Subject	Classification (%)								
	Arousal			Valence			4 Basic Emotion		
	LR	IWLR	SOMLR	LR	IWLR	SOMLR	LR	IWLR	SOMLR
1	95.8	91.7	89.2	78.1	78.1	73.3	77.3	78.7	69.3
2	87.5	93.3	97.5	81.0	75.2	74.3	82.7	77.3	73.3
Average	91.7	92.5	93.3	79.6	76.7	73.8	80.0	78.0	71.3

5.5 Conclusion

In this chapter, we proved that local learning can be used to address covariate shift. It is a post-learning approach, which modifies the prediction to adapt to covariate shift. This allows it to be able to adapt to changing covariate shift. Shifting the weighting from learning process to prediction process also allows local learning to perform online prediction faster, thus, making the method more suitable for applications that require timely responses.

Applying local learning methods in both inverse kinematic modeling and vocal emotion recognition shows that the local learning approach is as effective as importance-weighted learning in addressing a covariate shift issue. However, the weighting methods work well when there is a clear sign of covariate shift. There is no guarantee of improvement when there is none.

Chapter 6

Feature Augmentation: Converting Concept Shifts to Covariate Shifts

While importance-weighted learning and local learning methods can handle covariate shifts, they cannot handle concept shifts. This chapter introduces the feature augmentation technique to remedy this shortfall. By expanding the feature space to include recently matched features and labels, we can apply local learning to solve concept-shift problems. To prove the usefulness of this technique, we applied this technique to water shooting control. The work presented in this chapter has been published in [25].

6.1 Concept

As mentioned before, a covariate shift is an easier problem than a concept shift. Thus, it is beneficial to explore how a concept-shift problem can be converted into a covariate-shift problem. From the literature review in Chapter 1, we learned a class of concept-shift compensation methods that integrates contextual information into the features or input variables [11]. It is much simpler, and thus in this chapter, we develop a method that makes use of this concept.

In online supervised learning, we want to predict the output, $y \in \mathcal{R}$, based on its features or input variables, $\mathbf{x} \in \mathcal{R}^n, n \in \mathcal{Z}^+$. The prediction model, f is constructed from past $(\mathbf{x}, y)$ data pairs, such that

$$y = f(\mathbf{x}). \quad (6.1)$$

The model is then used to predict future y . A concept shift occurs when the relation between $\mathbf{x}$ and y changes over time [8], where the above model is no longer usable. Under this condition, a new mapping, g should be learned:

$$y = g(\mathbf{x}). \quad (6.2)$$

This forgetting mechanism is what most of the online learning methods employ as reviewed in Chapter 1. However, discarding old data in favor of the new data could be wasteful. What if the old condition recurs? It is inefficient to relearn the old concept that we have acquired before. Can we learn both models in Equation 6.1, 6.2, and possibly many other shifted models by using just a single prediction model?

Another perspective to look at a concept shift is missing co-variables or features. We can think of the original feature, $\mathbf{x}$, to be insufficient to fully predict y under changing condition. Another set of features, $\mathbf{x}'$, are needed to explain the condition change. If we can find such features, we can unify the many changing models into one model, h :

$$y = h(\mathbf{x}, \mathbf{x}'). \quad (6.3)$$

Alas, we generally do not know the full set of features in practice. This incomplete feature leaves us with a concept-shift problem.

Instead of looking for the missing information, we can make full use of currently available data. One example is to look at the recently mapped data. To illustrate this, two hypothetical mappings $y = f_1(x)$ and $y = f_2(x)$ corresponding to two different conditions, z_1 and z_2 , are shown in Figure 6.1. From the figure, we can see that for the same x_t , there are two possible predicted values, y_t^1 and y_t^2 . We are unable to determine which mapping to use without knowing the actual condition. Hence, it is impossible for us to predict the correct y based on x alone. However, if we know the recently mapped (x_{t-1}, y_{t-1}) , we can infer the current condition by assuming that the current condition does not change much from when the data pair was generated. For instance, if the recent x_{t-1} was mapped to y_{t-1}^1 as shown

FIGURE 6.1: Illustration of feature augmentation

in Figure 6.1, then the correct mapping is f_1 . Similarly, if it was mapped to y_{t-1}^2 , the correct mapping is f_2 .

The illustration in Figure 6.1 shows that we can implicitly infer the current condition by having learning algorithms trained to predict y based on not only current $\mathbf{x}$ but also paired $\mathbf{x}$ and y immediately before it. Therefore, we augment the input argument of the mapping to include the immediate past data pair:

$$y_t = f((\mathbf{x}_{t-1}, y_{t-1}), \mathbf{x}_t). \quad (6.4)$$

As a consequence, the change in mapping reflected on the recent data pair is now contained in the input argument. In other words, the change in mapping becomes a change in input distribution. The original concept-shift problem:

$$p(\mathbf{x}_t, y_t) = p(y_t | \mathbf{x}_t) p(\mathbf{x}_t), \quad (6.5)$$

where both the mapping $p(y_t | \mathbf{x}_t)$ and input distribution $p(\mathbf{x}_t)$ are varying, has become a covariate-shift problem:

$$p(\mathbf{x}_t, y_t, \mathbf{x}_{t-1}, y_{t-1}) = p(y_t | \mathbf{x}_t, \mathbf{x}_{t-1}, y_{t-1}) p(\mathbf{x}_t, \mathbf{x}_{t-1}, y_{t-1}), \quad (6.6)$$

where only the input distribution $p(\mathbf{x}_t, \mathbf{x}_{t-1}, y_{t-1})$ is varying.

6.2 Application: Water Shooting Control

6.2.1 Motivation

This work is motivated by the Fukushima nuclear plant meltdown incident in 2011. In this unfortunate event, to avoid exposure to radioactivity, the response team tried to cool down the exposed fuel rod by sending a helicopter to scoop up nearby sea water and release the water from a distance above the plant. Unfortunately, due to high winds and the motion of the helicopter itself, most of the water missed the target. Later fire trucks with water cannons were deployed to douse the reactor from a distance. Again, due to strong wind, the result was unsatisfactory. Therefore, we explore a machine controlled water cannon alternative that can perform water shooting from a distance more accurately without risking human operators.

There are two main challenges that machine controllers must overcome in the context of a water shooting scenario. The first challenge is the non-stationary condition, where the wind speed and direction vary over time. Besides, the water cannon position changes with time too. Consequently, machines need to sense regularly the changes and adjust the shooting angle accordingly. Second, the time it takes for the water to reach the target introduces a delay in feedback (time-delay system). Since conventional machine controllers such as PID rely heavily on instant error feedback to generate a corresponding reaction, the delay makes them slow to react to changes.

6.2.2 Literature Review

To design a solution for these two challenges, we studied learning-based controllers for their adaptability and ability to predict, the concept shift theory for its capacity to model non-stationary conditions, and time delay control. Concept-shift theory has been discussed extensively in Chapter 1, 3, and 5. We focus on the other two topics in this sub-section.

6.2.2.1 Learning-based Control

Learning-based control in robotics can be divided into model-based and model-free methods [4]. Model-based methods construct models that describe the behavior of a plant from data comprising inputs to the plant and the corresponding response, and use the models for control. Model-free methods directly learn how to control the plant without modeling its behavior [4]. In this application, a model-based method is used because the model can be utilized for internal simulations to predict the future behavior of a plant [68]. The ability of model-based methods to learn models from data can be leveraged in situations where the analytical model is hard to establish [35]. On top of that, using online supervised learning algorithms such as the receptive field-weighted regression (RFWR) [60] to learn models allows the controller to adapt to time-dependent changes [35]. In our context, the changes refer to concept shifts, which we discussed in Section 6.1.

6.2.2.2 Time-Delay Control

In control theory and engineering, the general analysis of time-delay systems focuses on the effect of time delays on the system stability [69] and dead time compensation (DTC) [70] with the assumption of constant and known time delays [71]. The Smith Predictor [72] is a standard DTC method. It uses an extra loop with a predicting function to predict the future feedback value. Next, the predicted feedback value is used as the feedback to a conventional controller. The predicted feedback allows control actions to be generated without waiting for the actual feedback that is delayed. Another assumption made in time-delay system control is that the predicting function is known beforehand, whereas in learning-based control the predicting function is learned from data.

The latest research in time delay system control involves delay identification and time-varying delay. Delay identification addresses the problem of unknown time delay. Correlation-based methods have long been used in the field of signal processing to estimate time delays, while in the field of control, steepest descent-based algorithms are used [71]. The research on time-varying delay focuses on robust stability under changing time delay [73]. In online learning, time delay is often assumed not fixed but known with time-stamped feedback [74, 75].

The nature of the time delay in the water shooting problem is time varying and unknown given the non-stationary conditions. Our solution, the Model-based Learning Controller, which is to be described in the next sub-section, galvanizes the three fields: control, signal processing, and machine learning to learn a predicting function (Smith Predictor) from feedback data with unknown and time-varying delays matched using a cross-correlation-based method.

6.2.3 Model-based Learning Controller

A model-based learning controller (MBLC) consists of two components, namely, model learning and control policy. Based on feedback from a sensing system, a control model matching the input and output is constructed using a regression algorithm. As mentioned before, to adapt to time-dependent changes, online regression algorithms such as RFWR can be employed. The constructed model is then used to generate control signals by a handcrafted control policy.

The controller is constructed by integrating (i) feature augmentation (FA), (ii) delay estimation and data matching, and (iii) prediction model (PM). Together they help to address the non-stationary conditions and time delay challenges. Figure 6.2 shows the overview of the proposed control system.

FIGURE 6.2: Overview of the proposed model-based learning controller (MBLC)

6.2.3.1 Water-Shooting Model

In the given water-shooting problem, our aim is to adjust the shooting angle so that the water reaches the target under changing wind conditions and vehicle motion. In other words, given the (i) wind condition, $\mathbf{u}(t)$ from the shooting time, t_s , till the time it hits an object, $t_s + \tau$, (ii) the vehicle motion at the shooting time, $\mathbf{v}$, and (iii) the desired final position, $\mathbf{x}$, we want to determine the correct shooting angle, θ . This question can be posed as a function mapping problem:

$$\theta_{t_s} = f(\mathbf{x}, \mathbf{v}_{t_s}, \mathbf{u}(t)_{t_s}^{t_s+\tau}). \quad (6.7)$$

To determine the mapping, we use a simple 2D model to simulate water shooting (Figure 6.3). The model consists of: water, water cannon, vehicle, target, wind, and forces. The water is modeled as a stream of solid spheres. The water cannon with a body length l is mounted on a movable vehicle and has a variable shooting angle, θ . These variables determine the initial position of the water spheres, $\mathbf{x}_{t_s}$. The water cannon tilts at a constant speed and shoots water at a constant flow rate, which results in a constant initial water speed, $\dot{\mathbf{x}}_{t_s}$. The target is located at some distance and height (d, h) , away from the water cannon.

FIGURE 6.3: Water shooting simulation

A 2D wind, $\mathbf{u}(t)$, which varies its strength and direction over time, is introduced in the model. The wind incurs a drag force on the water spheres with its magnitude

proportional to the relative velocity, $\mathbf{u}(t) - \dot{\mathbf{x}}(t)$. Lastly, a gravity field, $\mathbf{g}$ is added to the model. Based on this model, we derive the following equation that gives the final position of a water sphere:

$$\mathbf{x} = \mathbf{x}_{t_s} + \dot{\mathbf{x}}_{t_s} \tau + \frac{1}{2} \mathbf{g} \tau^2 + \frac{3}{8} \frac{\rho_{air}}{\rho_{water}} \frac{C_d}{r} \int_{t_s}^{t_s+\tau} \int_{t_s}^{t_s+\tau} \|\mathbf{u}(t) - \dot{\mathbf{x}}(t)\| (\mathbf{u}(t) - \dot{\mathbf{x}}(t)) dt^2, \quad (6.8)$$

where ρ_{air}/ρ_{water} is the density ratio between air and water, r is the radius of the water sphere, and C_d is the drag coefficient.

Though the model is simplistic, it captures the non-stationary condition and time delay aspect of the water shooting problem. The non-stationary condition is caused by the changing wind and vehicle motion. The vehicle motion affects the initial position, $\mathbf{x}_{t_s}$, and velocity, $\dot{\mathbf{x}}_{t_s}$, and subsequently, the final position, $\mathbf{x}$. The wind has an accumulated effect from t_s to $t_s + \tau$ on the final position as given by the double integration in Equation 6.8. Equation 6.8 also indicates that the action (the shooting angle θ , which affects the initial position, $\mathbf{x}_{t_s}$) and the effect (the final hit position, $\mathbf{x}$) occur at t_s and $t_s + \tau$, respectively. This difference in time results in a time delay, τ , in the feedback.

Deriving the inverse mapping in the form of Equation 6.7 from Equation 6.8 is difficult because of the time-varying delay caused by the changing wind and vehicle motion. We thus propose using machine learning techniques to construct the inverse mapping from data.

6.2.3.2 Feature Augmentation for Non-Stationary Condition

For simplicity, this sub-section explains the construction of the inverse mapping under changing wind condition with zero vehicle motion and no time delay in feedback. Assuming that we can measure the hit position of water, $\mathbf{x}$, its initial shooting angle, θ , and the wind condition, $\mathbf{u}(t)$, we can collect these data over time and perform regression to construct the above inverse function as the control model. However, it is not easy to measure the time- and space-varying wind condition. This hurdle leaves us with only the shooting angle and hit position data. The mapping becomes:

$$\theta = f_{\mathbf{u}(t)}(\mathbf{x}). \quad (6.9)$$

The function, $f_{\mathbf{u}(t)}$, now becomes variable, depending on the wind condition, $\mathbf{u}(t)$. Thus, the use of shooting angle and hit position data alone to construct the model is not feasible. This new mapping can be considered a concept-shift scenario, where the relationship between input and output, $p(\mathbf{x}|\theta)$, changes with time due to the change in wind conditions over time.

In the literature review, we mentioned that the active concept-shift compensation methods make use of recent data to construct a new model to reflect the current situation. The mechanism makes sense because the adjacent data can be assumed to originate from the same condition as the current one. Using the same idea, we can use the last known shooting angle and hit position, $(\mathbf{x}_{t-1}, \theta_{t-1})$, to infer the wind condition that is going to affect the current hit position, $\mathbf{x}_t$. This is done by augmenting the input argument of the mapping to include the last known shooting angle and hit position:

$$\theta_t = f((\mathbf{x}_{t-1}, \theta_{t-1}), \mathbf{x}_t) \quad (6.10)$$

as indicated by the extra matched data pair in Figure 6.2. This turns the original concept-shift problem:

$$p(\mathbf{x}_t, \theta_t) = p(\mathbf{x}_t|\theta_t) p(\theta_t), \quad (6.11)$$

where both the mapping $p(\mathbf{x}_t|\theta_t)$ and input distribution $p(\theta_t)$ are varying, into a covariate-shift problem:

$$p(\mathbf{x}_t, \theta_t, \mathbf{x}_{t-1}, \theta_{t-1}) = p(\mathbf{x}_t|\theta_t, \mathbf{x}_{t-1}, \theta_{t-1}) p(\theta_t, \mathbf{x}_{t-1}, \theta_{t-1}), \quad (6.12)$$

where only the input distribution $p(\theta_t, \mathbf{x}_{t-1}, \theta_{t-1})$ is varying.

6.2.3.3 Online Regression Algorithm

For covariate-shift problems, we have discussed in depth in Chapter 5 that local learning algorithms are suitable as they confine the learning into small regions of an input space. They help to achieve a better bias-variance trade-off by adapting model parameters according to local regions [57]. They are also used to limit the generalization capability of a local model to avoid negative interference or catastrophic forgetting in incremental learning [58]. The latter feature is advantageous for online learning with virtual concept drift. Since the water shooting problem involves only 3-dimensional data, the local learning algorithm we used is Receptive

Field-Weighted Regression (RFWR) algorithm [60]. It constructs multiple independent local regression models from data and predicts by combining weighted prediction from all the models. The weighting is based on the activation strength of the local models measured by a predefined kernel function. For problems with higher dimension, Locally Weighted Projection Regression (LWPR) algorithm [76] can be employed.

6.2.3.4 Prediction Model to Address Time Delay

If we include the time-delay effect, Equation 6.10 becomes:

$$\theta_t = f((\mathbf{x}_t, \theta_{t-\tau(t)}), \mathbf{x}_{t+\tau_1}). \quad (6.13)$$

The equation means that the desired final hit position, $\mathbf{x}_{t+\tau_1}$, at time $t + \tau_1$ (in the future after a delay, τ_1) could be achieved by shooting at angle θ at the current time, t . The prediction is based on the current feedback of the hit position at time t and its corresponding shooting angle at time $\tau(t)$ earlier. Though the received mapping, $(\mathbf{x}_t, \theta_{t-\tau(t)})$, reflects the current condition, it might not reflect the future condition, between current time t and $t + \tau_1$. If the condition changes within this period, the predicted shooting angle, θ_t , will not be able to produce the desired hit position, $\mathbf{x}_{t+\tau_1}$. The longer the delay, τ_1 , the more likely the change occurs. To avoid the uncertainty within this time gap, we could reduce the time gap by using the mapping immediately before the expected future time:

$$\theta_t = f((\mathbf{x}_{t-1+\tau_2}, \theta_{t-1}), \mathbf{x}_{t+\tau_1}). \quad (6.14)$$

However, we only know the shooting angle immediately before, θ_{t-1} . The corresponding hit position, $\mathbf{x}_{t-1+\tau_2}$, is yet to be received. If the change in condition exhibits recurring patterns like a sinusoidal change, we can predict the future hit position by learning the following function mapping:

$$\mathbf{x}_{t-1+\tau_2} = f((\mathbf{x}_t, \theta_{t-\tau(t)}), \theta_{t-1}). \quad (6.15)$$

Unlike Equation 6.14, Equation 6.15 is a many-to-one mapping, which is relatively easier to learn. Hence, we can use a regression algorithm, such as RFWR to learn this mapping.

Similar to Equation 6.13, Equation 6.15 has a time gap issue. To transform Equation 6.15 to the form similar to Equation 6.14 to reduce the time gap, we perform a series of step-by-step prediction as shown below:

$$\begin{aligned}
 \mathbf{x}_{t+1} &= f((\mathbf{x}_t, \theta_{t-\tau(t)}), \theta_{t-\tau(t)+1}); \\
 \mathbf{x}_{t+2} &= f((\mathbf{x}_{t+1}, \theta_{t-\tau(t)+1}), \theta_{t-\tau(t)+2}); \\
 &\vdots \\
 \mathbf{x}_{t-1+\tau_2} &= f((\mathbf{x}_{t-2+\tau_3}, \theta_{t-2}), \theta_{t-1}).
 \end{aligned} \tag{6.16}$$

The first prediction makes use of existing information $(\mathbf{x}_t, \theta_{t-\tau(t)}, \theta_{t+1-\tau(t)})$ while the subsequent predictions are made based on the previous prediction.

Using the Smith Predictor as an analogy, Equation 6.16 serves as the internal prediction model (g in Figure 6.2) to predict the delay-free response, which is then sent to the controller as the feedback in Equation 6.14. Since the predicted response is delay-free, we can be sure that the action generated by the controller can catch up with the changes in condition. In this way, we address the time-delay issue.

6.2.3.5 Time Delay Estimation to Map Delayed Feedbacks

In constructing the training data used by the regression algorithm to learn the function mapping in Equation 6.14 and Equation 6.16, we need to measure the time delay to match input data points with their corresponding delayed output data points as shown in the second component in Figure 6.2. The measurement involves tracking water jet from the nozzle to the target, which is more challenging than measuring the hit position.

Alternatively, we can make an estimate by interpolating empirical time-delay data collected from a series of controlled water-shooting tests. The tests are conducted using a set of predetermined target positions. With a constant condition, the delay time for each target position is measured. With these target positions and the corresponding time delays, we estimate the time delay of other positions by interpolation. Due to the assumption of constant condition, the estimation error by this interpolation-based method could be high.

A third approach is to use a signal processing technique called cross-correlation. Since we have continuous access to the input data and output data, we can treat them as signals and match them. The matching is done by shifting a window of the output signal backward along the temporal axis until the signal profile matches with a profile of a segment of the input signal. The amount of shift tells us the time delay. This method does not assume a constant condition. Hence, it can give a more accurate time delay estimate. As both input and output signals are of different magnitude, we utilize the normalized version of cross correlation:

$$(\overline{f * g})[n] = \sum_{m=1}^M \frac{(f[m] - \bar{f})(g[m+n] - \bar{g})}{\sigma_f \sigma_g}, \quad (6.17)$$

where f and g are the two signals with a window length of M ; m is the index of the signals; $\bar{f}$, $\bar{g}$, σ_f^2 and σ_g^2 are their mean and variance; n is the index of the cross-correlation values, $(\overline{f * g})$, ranging from $-M + 1$ to $M - 1$. To determine the time delay, we find the index that corresponds to the maximum cross-correlation value:

$$\tau = dt \cdot \arg \max_n (\overline{f * g})[n]. \quad (6.18)$$

Ideally, a perfect match should result in a maximum value of 1.0. However, the variable time delay distorts the profile of the delayed output, making the maximum value smaller than 1.0. Despite that, the value should be close to 1.0. To avoid a dissimilar match, a threshold is used. If the maximum value is below this threshold, no time-delay estimation is performed.

In this method, we assume that there is a unique match for each output profile. This assumption is true if the controllable input has a greater influence on the output than the uncontrollable changing condition. Also, if the input signal is constant, the cross-correlation-based method will not work because there are no salient features in the input signal for matching and multiple matches are possible. Under this condition, the time delay cannot be estimated.

To overcome this problem, we introduce probing inputs in the form of random spikes to add salient features to both input and output signals. Unfortunately, we cannot use the probing input all the time, or it will interrupt the control process. We can only use them at fixed intervals. This interval also gives us a reference to set the size of the cross-correlation window; the window size should be bigger than

the probing signal interval. This requirement ensures at least one probing signal appears in a cross-correlation window.

Even with the probing signals, the cross-correlation-based method does not guarantee that a time delay can be estimated. To complement this shortage, we combine both interpolation-based and cross-correlation-based time-delay estimation methods. The combination is carried out such that whenever possible, the time delay is estimated by the cross-correlation-based method. When it is not feasible, we switch to the interpolation-based method. To compensate for the estimation error by the interpolation method, we determine the difference between the two estimates:

$$e_{\tau}(t) = \tau_{xcorr}(t) - \tau_{interpolation}(t). \quad (6.19)$$

The average of the recent d differences can then be added to the time delay estimated by the interpolation-based method when the cross-correlation estimation is not available:

$$\tau = \tau_{interpolation}(t) + \frac{1}{d} \sum_{t=T-d+1}^T e_{\tau}(t). \quad (6.20)$$

In this way, we reduce the time-delay error caused by ignoring the condition change.

6.2.3.6 Control Policy

To incorporate the active probing for time delay estimation, we decided not to perform control policy learning. Instead, we created a finite state machine (FSM) to encode the control policy (Figure 6.4). This FSM also allows us to perform the control straightaway instead of requiring repeated trial and error in learning the control policy.

As the control model is constructed from data, the prediction could be erratic if there are insufficient data. This situation often happens during the initial phase, or learning phase, of the control task. To overcome this shortage, we introduce the first state (State 0) that ensures that the water cannon reaches its target quickly. This is achieved by incorporating a simplified version of Equation 6.8 that ignores

FIGURE 6.4: Finite state machine for exploration and quick initial prediction

the wind effect as shown below:

$$\begin{aligned} \mathbf{x} &= \mathbf{x}_{t_s} + \dot{\mathbf{x}}_{t_s} \tau + \frac{1}{2} \mathbf{g} \tau^2, \\ \tau &= \frac{X - x(0)}{\dot{x}(0)}, \end{aligned} \quad (6.21)$$

where τ is the delay estimate calculated from the horizontal distance of a target, X ; $x(0)$ is the initial nozzle x -position; and $\dot{x}$ is the initial horizontal exit velocity of the water jet. Without considering the wind effect, the simplified model provides a quick prediction of shooting angle but with low accuracy.

After State 0, the water cannon stays at the predicted shooting angle for τ_1 number of error feedback (State 1). This state is necessary because water needs time to travel and hence there is a delay before a feedback can be received. This state prevents the water cannon from blindly moving before it perceives the actual situation through error feedback. The time delay τ_1 is set based on the dimension of the input variables. In our case, the dimension is three as indicated by Equation 6.14 and Equation 6.16. Hence, we set the delay to be three. The first two states constitute the initial learning phase that allows the controller to collect sufficient data to build the control models.

Another purpose of this finite state machine is to enable the controller to explore

different regions in a task space for data collection (State 2) and to mitigate the effect of erratic prediction such as no prediction or out-of-range prediction (State 3). The state machine checks if a prediction given by the learned control model is within the shooting angle range. An erratic prediction indicates that the learned control model does not have sufficient data for accurate prediction. Then the finite state machine reverses the moving direction and starts an exploration scheme so that more data can be collected, transitioning from State 3 to 2. Otherwise, the controller moves the water cannon to the predicted angle in State 4.

The cycle of Explore-Predict-Move continues throughout the control process. Through this cycle, the learning machine collects new data and updates its control model when a novel wind condition is encountered. In this way, it keeps the control model up-to-date and accurate.

At fixed interval t_p , the whole process is interrupted by State 5, where the water cannon moves according to the randomly selected probing signal. After the probing motion is executed, the controller returns to the state before State 5. The introduction of State 5 is for the purpose of time-delay estimation as discussed in the previous subsection.

6.2.4 Experiments

In this subsection, we describe the implementation of the machine learning-based controller and compare its performance in shooting water under discrete and continuous condition changes with human operators and a PID controller.

6.2.4.1 Computer Simulation Model

To show the possible applications of the proposed controller, we model the simulation after two scenarios: a high rise building fire (Figure 6.5) and an oil rig fire (Figure 6.6). In the first scenario, the water cannon is mounted on a stationary truck, and the target or fire can be at any of the 56 windows of a 25 m tall building. The source of the non-stationary condition is the sudden wind change, either in direction or magnitude, to simulate wind gusts. In the case of the oil rig fire, the target can be anywhere on the oil rig, and the water cannon is mounted on

FIGURE 6.5: Building Fire simulation

FIGURE 6.6: Oil rig fire simulation

a fire boat which moves with ocean waves. The wave movement is modeled as a continuous sinusoidal vertical and pitch motion.

In both cases, a green vertical bar is used to mark the target position and the tolerance. As this study focuses on the effect of non-stationary conditions and time delay, we eliminate other confounding factors such as increased sensitivity to angle deviation with a larger distance by using different tolerances at different locations. In this way, regardless of the distance, the same angle deviation from all targets always results in a same change in the hitting rate.

A software sensor is included in the simulation to track the error of water spheres hitting on the building or oil rig and calculate the hit score. It generates error feedback with unknown delay (0.3 to 1.5 seconds). Gaussian noise ($\mu = 0$, $\sigma = 0.03$) is added to the feedback error but mitigated by a windowed averaging filter. In real world situations, we can use a camera with computer vision to extract the water hit position.

To depict the real situation as closely as possible, we used actual metric values (Table 6.1). For example, the usual water cannon range is about 40 meters, so the initial velocity of the water spheres is set such that the water can reach the distance under no wind condition. The wind speed range used in the simulation is obtained from actual meteorological data around Pittsburgh, PA in June 2014 because of data availability. The data recorded 2.5 m/s average wind speed, an average gust of 11.0 m/s and a maximum gust of 22 m/s.

TABLE 6.1: Simulation Model Specifications

Components	Specification	Values
Target	Height, h	0 to 25 m
	Distance, d	7 to 23 m
Water	Radius, r	0.05 m
	Initial speed, v_0	25 m/s
Water Cannon	Length, l	1 m
	Shooting angle, θ	-5 to 70 degree
	Rotational speed, ω	20 degree/s
Wind	Drag coefficient, C_d	0.47
	Wind Speed, u	0 to 25 m/s
Vehicle	Vertical motion magnitude, v	0.1 m
	Rotation magnitude, α	0.05 radian
	Period, T	8 s

6.2.4.2 Time Delay Matching

For the interpolation-based time-delay estimation, we performed simulations based on Equation 6.8 to obtain the simulated time delay values for a grid of target positions. Three probing signals are used for the cross-correlation-based method in random order at fixed interval during the simulation. They are an upward spike, downward spike, and the combination of the two. Table 6.2 shows the values used in estimating the time delay for matching the input and output data points.

TABLE 6.2: Parameters for Time Delay Estimation

Components	Specification	Values
Interpolation	Number of grid point, n_{gp}	27×26 m
	Grid size, Δ_g	1 m
	Horizontal range, r_x	2 to 27 m
	Vertical range, r_y	1 to 27 m
	Number of past error, n_e	5
Cross-correlation	Window size, M	50
	Threshold, th_{xcorr}	0.45
Probing signals	Magnitude, Δ_{ps}	$\omega \cdot dt$
	Interval, t_p	0.5 s

For the high-rise building fire, a combination of the interpolation-based and cross-correlation-based method is used because the condition is constant most of the time. For the oil rig fire, the active probing method is not suitable as the controller needs to adjust to changes continuously. Hence, we implemented the interpolation-based method.

6.2.4.3 User Interface for Human Operators

We created an input interface to allow human operators to control the shooting angle of the water cannon using a keyboard. Human operators pressed UP and DOWN key for fine increment or decrement of the shooting angle. For a coarse and speedy change of the angle, human operators could press PAGEUP and PAGE-DOWN key. This combination allows both precision and speed.

6.2.4.4 Model-based Learning Controller vs. Human Operators

We compared the performance of the proposed controller and human operators using the computer simulation to determine if an MBLC could replace a human operator for a water shooting task under changing conditions.

There were two tests to compare the performance of the proposed controller and human operators. The first test used the high-rise building fire scenario with sudden wind change, while the second test used the oil rig fire scenario with continuous boat motion. For each test, two different target locations were tested. Location 1 had a shorter distance, while Location 2 had a longer distance.

For the first test, both the human operators and MBLC were allowed three trials (each trial lasts 30 secs) to improve the performance. In each trial, the wind condition was slightly different in sequence and duration. The slight change is to prevent rote learning. For the second test, instead of three trials, prolonged test duration of 90s was used. Figure 6.7 (a) and (b) show an example of the wind change and vehicle motion change, respectively. The other details of the experiment are summarized in Table 6.3.

FIGURE 6.7: Non-stationary conditions: (a) Wind gust change as the discrete non-stationary conditions; (b) Rocking boat as the continuous non-stationary conditions.

Twelve engineering students were employed as the human operators. In order to familiarize themselves with the interface and the computer simulation, they were allowed unlimited time to practice before the tests. The practice sessions consisted of wind types and fire locations different from the actual tests to prevent them from

TABLE 6.3: Man vs. Machine Experiment Details

Scenario	Target	Target Tolerance	No. Trials	Duration
Sudden wind change	(16.2, 18.5) m	± 0.50 m	3	30 s
	(20.7, 21.0) m	± 0.55 m	3	30 s
Continuous boat motion	(10.0, 12.5) m	± 0.32 m	1	90 s
	(17.0, 15.5) m	± 0.43 m	1	90 s

memorizing the test conditions. In between tests, human operators were allowed to take a break to avoid fatigue or strain.

6.2.4.5 Model-based Learning Controller vs. PID Controller

The MBLC was benchmarked against a conventional PID controller. We used the same two scenarios but increase the number of locations (55 Windows for the wind gust scenario and 17 locations for the rocking boat scenario).

For faster initial response without waiting for delayed error feedback, the PID controller is modified such that the shooting angle at the beginning is controlled by a pre-defined model similar to the one used in the MBLC. The use of this fixed model allows for direct comparison of the two controllers. The initial shooting angle was set at 45 degrees, which is different from the required shooting angle. The PID gains were tuned using the target location at (9.25, 21.00) m and process is as follows: First, the proportional gain was tuned from 0.5 down to 0.022 until the step response is not oscillatory but not sluggish either, with slight overshoot. This tuning usually results in a better hitting rate. A derivative gain was introduced to damp the overshoot. It was tuned such that the score improves, giving a final gain of 0.5. The introduction of non-zero integral gain gave an oscillatory response. Hence, the integral gain was set to zero.

6.2.5 Result and Discussion

6.2.5.1 Model-based Learning Controller vs. Human Operators

Figure 6.8 compares the human operators and MBLC. Comparing the performance at Location 1 and 2, we can see a clear decrease in performance for all the controllers and scenarios. This decrease is because Location 2 incurs a longer delay,

FIGURE 6.8: Comparison between human and model-based learning controllers under sudden wind change and continuous boat motion at two locations.

which affects both the human operators and machine controller. Regardless of the target location and scenarios, we observe that the performance of both the human operators and MBLC improve over time. This improvement shows that both the human operators and controller are capable of learning from their experience to improve their performance. Comparing the two, we can see that the MBLC performs on average better than all the human operators for both locations and scenarios.

6.2.5.2 Model-based Learning Controller vs. PID Controller

Figure 6.9 compares the PID and MBLC. For the discrete case, the MBLC (red curve) performed slightly better than the PID controller (blue curve). The performance gap is even wider when the PID controller was not aided by the initial model (green curve). On the other hand, the PID controller performed much worse than the MBLC in the case of the continuous change in condition. For both the discrete

FIGURE 6.9: Comparison between PID and model-based learning controllers under sudden wind change and continuous boat motion at two locations

and continuous cases, the MBLC's performance increased when we improved the accuracy of the time-delay estimation (magenta curves).

6.2.5.3 Discussion

With the feature augmentation, the MBLC can adapt to non-stationary conditions, be it a discrete or continuous change. Figure 6.10 (a) and (b) shows how the controller reacted with changes. At every 5s when the wind changed, we see in the first plot that the error deviated from zero (blue curve). Subsequently, the controller changed the shooting angle (red curve), which shifted the error back to near zero. Similarly, for the continuous change in the second plot, the controller moved the cannon continuously (red curve) such that the error (blue curve) was kept within the tolerance (dashed horizontal lines).

The downside of feature augmentation is its increase in dimension. For higher dimensional problems, regression methods with dimensionality reduction such as LWPR are necessary.

The use of the prediction model helps to improve the water-shooting performance in the rocking boat scenario, where the error (error plot in Figure 6.11 (a)) generated by the controller with the prediction model (red curve) is smaller than the one produced by the controller alone (blue curve). This improvement is because the

FIGURE 6.10: How model-based learning controller react to change

controller with the prediction model can preemptively react as shown in the phase-lead in the controller's shooting angle (angle plot in Figure 6.11 (b)). Note that the prediction model is purely for time-delay compensation, which is different from model predictive control techniques that try to optimize the current action for a short time horizon in the future.

FIGURE 6.11: Improvement due to the prediction model: (a) Error reduction
(b) pitch angle change

The time delay estimation process affects the performance of the controller (Figure 6.9) because it provides matched data to the model learning process. The accuracy of the time delay estimation impacts the quality of the data and subsequently the accuracy of the models.

6.3 Conclusion

In the previous section, we automate water shooting using a novel model-based learning controller. The controller can address the non-stationary condition and time delay using the feature augmentation, prediction model, and time-delay estimation. The controller performs better than both the human operators and the PID controller in handling both the sudden and continuous changes in condition.

The water shooting application shows that we can expand the feature space of the prediction model to address concept shifts. This post-learning method combines feature augmentation with local learning, and thus it can handle frequent changes in conditions.

Part III

Pre-Learning Compensation

Chapter 7

Condition-Specific Learning: Avoiding Non-Stationary Condition

This chapter presents an example of pre-learning techniques that involves data preparation. The idea is simple: totally avoiding non-stationary conditions by making sure both the training data and test data are generated from the same source. We employ this technique to develop a semi-automatic snore labeling software, which requires only minimal human inputs to label a one-night-long snore recording. The work presented in this chapter has been published in [26].

7.1 Concept

Datasets used for machine learning usually contain multiple sources of variation [77]. Depending on the objective we want to achieve, we choose the sources of variation we want to distinguish. In supervised learning, the objective is usually represented by data labels, while the features extracted from data should ideally capture the relevant sources of variation. However, features used may not cover all the sources of variation, or even the sources of variation relevant to our objective. In some instances, sources of variation that are specific to only non-general conditions might

be selected due to the lack of comprehensive training data. This makes the resulting classifier fail to generalize to other data that do not share the specific source of variation.

The problem is further complicated when we pool data from multiple conditions. Multiple data sources are used because of insufficient data from each individual source, or we want to search for features that are generic and robust across multiple conditions. This practice introduces more sources of variation, including those not relevant to the learning task at hand. In this scenario, careful feature engineering is necessary. However, feature engineering is difficult, be it in selecting relevant features [78], or constructing features based on field knowledge [28].

Given the difficulty in identifying relevant sources of variation, we can try to minimize the difference in sources of variation between the training and test condition by using data from a single source. This can be achieved only when we have abundant data from the same source. In this setting, we can construct a specific model trained and tested on data that originated from a single source. Since the model trained will be used only for data from the same source, we do not need to worry if the model trained can be generalized to other conditions. We call this method condition-specific learning.

Snore detection discussed in the next section is one of such rare applications where abundant data from a single source are available. For each subject, there is a one-night-long (6-8 hours) audio recording.

7.2 Application: Semi-automatic Snore Labeling

7.2.1 Motivation

Humans spend one-third of their lives sleeping. The quality of sleep affects not only the performance on the next day but also one's general health. One of the most common sleep-related issues is snoring. Research has shown that snoring can be an indicator of other health problems. For example, snoring is an important marker for obstructive sleep apnea (OSA). It is also associated with increased coronary heart disease and stroke in women [79], myocardial infarction [80] and ischemic cerebral

infarction [81–83]. Another study even suggested that the vibration caused by snoring could damage nearby tissues [84].

These studies were based on self-reported snoring, which could be subjective [85]. To better study the relationship between snoring and health problems, more objective measure is required. The current practice relies on visual scoring by human experts to quantify snoring. It requires human experts to check through multiple full-night polysomnography (PSG) channels to identify snore events [86]. Its labor intensive nature prevents the extensive use of snore quantification in sleep and health studies. Also, it is insufficient just to detect the presence of snore. Intensity, duration, and types of snore often reveal more information about one’s health [86].

In conclusion, there is a need for a computational tool to help sleep researchers to better quantify snoring. The quantification could be useful in studying the correlation between one’s snore pattern and his or her health condition.

7.2.2 Related Works

Many attempts to use supervised learning methods to detect snores in audio signals automatically were reported. One of the early attempts is by Cavusoglu et al. [87], who used sub-band energy distributions with principal component analysis (PCA) and linear regression. Karunajeewa et al. [88] used richer features (number of zero crossings, energy of the signal, normalized auto-correlation coefficient at 1 ms delay, and first predictor coefficient of linear predictive coding analysis) with noise reduction and minimum-probability-of-error decision rule. Besides selecting the most effective features, researchers also studied the placement of microphones and found that tracheal recording is the most effective [89]. A recent study has shown more sophistication by further classifying snore sounds into four types: velum, oropharyngeal lateral walls, tongue base, and epiglottis [90].

The downside of supervised learning is that human annotation is still necessary. The quality of the method increases when more data are available. Again, the laborious nature of the human annotation limits the size of the dataset. In addition to that, there exists variation in snore features between subjects. The implication is that for each new subject, a new data set may be needed. This prevents the extensive use of this method.

To overcome these issues, researchers had turned to unsupervised learning. Azarbarzin et al. [91] combined V-Box algorithm and K-means algorithm to classify snore sounds based on sub-band frequency features with PCA. This method was further improved with the use of K-harmonics means clustering, which is not sensitive to initialization problem [92]. For easy clustering and visualization, dimension reduction techniques such as principal component analysis (PCA) were used. The use of reduced dimension reduces the explainability of the method. This causes hesitance and resistance in the medical field to employ this technology. In addition, clustering performed are usually binary. This does not help to identify different type of snores. Lastly, the methods above can only provide clustering. It is not clear to the authors how to identify a cluster as a snore cluster without knowing the labels beforehand.

In terms of data collection, all the methods mentioned above set up either a tracheal microphones, ambient microphones, or both to record sleep sounds at a sampling rate of 10kHz and above. The accuracy reported is high (about 90%). However, these techniques might not applicable to PSG data. In sleep laboratories or clinics, piezoelectric snore sensors or nasal cannula are more common, and usually, a low sampling rate (32-1024 Hz) is used. For instance, SWAN dataset uses 64 Hz [93]; MESA dataset uses 32 Hz [94], and Cleveland Family Study uses 256 Hz [95]. Though PSG data does not provide high-quality snore recording, the quantity of data available for analysis is far larger and covers a larger range of subjects with different health conditions as sleep clinics or laboratories throughout the world employ PSG. Hence, to further study the relationship between snoring and health condition, the use of PSG data is advantageous.

7.2.3 Overview

In this section, we attempt to bridge these gaps by automating the snore identification in PSG data. We employ unsupervised learning to avoid the reliance on human-annotated data. Sidestepping human annotation also allows us to use subject-specific data and subsequently avoid the variation in features between subjects to improve the classification accuracy. Different from the previous attempts, we chose to use hierarchical clustering in the hope that it can help us to differentiate different types of snores and improve the explainability of the classification results.

This not only helps the health professional to understand the snoring events better but also improves the acceptance rate of this technology in the field.

We describe our proposed algorithm in the following sub-section. In the Experiment sub-section, we detail our experiment to verify the proposed method. The last sub-section discusses the results of this study.

7.2.4 Snore Labeling Algorithm

We have developed an algorithm to label semi-automatically a full night (7 to 9 hours) of snore recording. A human user only needs to listen to a few clusters of random 5s auralized signal segments consisting of both snore and non-snore events and labels the segments accordingly. The labels are then used by the algorithm to automatically label the rest of the snore recording as shown in Figure 7.1.

Figure 7.2 shows the internal working of the algorithm. The algorithm utilizes both the signal processing and machine learning techniques. It consists of six steps, and only one of them requires inputs from a human user. The algorithm first extracts five-second segments which contain significant change in audio signal (sound). These segments are more likely to contain snores. Next, uniform sampling is performed to obtain ten percent of the total number of segments. Ten percent is used for convenience as long as there are sufficient samples (around nine samples for each cluster) for clustering in the later stage. Feature extraction is performed to extract the energy profile and the peak frequency of the signal. Then, normalized cross-correlation is performed between the segments' energy profiles. Together with the peak frequency values, the normalized cross-correlation values are used to perform hierarchical clustering. Human operators then label the resulting clusters. Lastly, the remaining 90% of the segments are classified based on the labeled clusters. The subsequent sub-sections discuss these steps in detail.

7.2.4.1 Signal Segmentation

In order to perform signal processing and clustering, we need to split the whole night signal into more manageable chunks. The naive method is to evenly divide the whole night signal. However, this might be a waste of computation if a subject

FIGURE 7.1: (a) Input: whole night of snore recording; (b) Output: labeled snore signal; (c) Snore signal (zoomed in); (d) Labeled snore signal (zoomed in)

only snores for a short period. Hence, we should only extract segments that are most likely to contain snore events.

Vertical Box (V-Box) algorithm [96] is employed to detect five-second segments that contain possible snore signals. The algorithm uses a box with a defined height, H and length, L to slide through the audio signal. It detects a change when the number of signal points contained within the box, b_{LH} falls below a threshold, $\theta \times L$. The dimension of the box and the threshold define what is considered a significant of change in the signal. The parameters used are: $L = 51$, $H = 5$, and $\theta = 0.95$, and they are determined empirically.

After the change markers are created by the V-Box algorithm, we segment the signal around the markers. Figure 7.3 illustrates these processes.

FIGURE 7.2: Six-step semi-automatic snore labeling algorithm

FIGURE 7.3: V-Box segmentation process (from top to bottom): input signal; count per box; change markers; merged episodes and segment centers marked by red lines

The segment size is chosen to be five seconds based on the range of respiratory rate from three to five seconds. Close-by change markers are merged into an episode. Episodes with a size smaller or equal to the segment size are extracted as segments, while episodes with a duration longer than five seconds are divided into multiple segments. For the latter case, we allowed the segments to overlap by 2.5 seconds to capture snore signals that happen to fall on the edge of the segments.

Note that the V-Box algorithm captures not only snore signals but also other non-snore but significant signal change. The two plots at the top of Figure 7.4 show two sample segments. The first one contains a snore event while the second one contains only noise.

7.2.4.2 Uniform Sampling

The main idea of this algorithm is to make use of small amount of samples labeled by a human user to label the majority of segments. For the samples to be representative, we perform uniform sampling to extract 10% of the segments generated from the previous step. Note that the sampling is done on the extracted segments and not on the whole-night signal. The value of 10% is chosen for convenience at the beginning. However, later analysis in Sub-Section 7.2.5 shows that 10% of the

FIGURE 7.4: Feature extraction: input raw signal, energy profile, and power spectrum with prominent peak frequencies for a snore segment (first column) and a noise segment (second column)

extracted segments for each subject contain sufficient segments for further processing. The remaining 90% will be labeled automatically by the classifier constructed by the 10% segments.

7.2.4.3 Feature Extraction

To illustrate the effectiveness of this algorithm and to maintain its explainability, we use only two simple features: energy and peak frequency. More advanced frequency-based features such as Mel-Frequency Cepstral Coefficients (MFCC) [92] or 500 Hz sub-band frequency distribution [91] are not used because, unlike audio recording, snore channels from PSG data are often sampled at a low frequency ranging from 64 to 1024 Hz. Moreover, these features are high-dimensional which requires dimension reduction. This reduces the explainability of the classification result.

The energy, E_s at point n is calculated as the mean of the square of the signal values within an N long window centered at n :

$$E_s[n] = \frac{1}{N} \sum_{i=n-N+1}^n |f[i; n]|^2. \quad (7.1)$$

The window size used in the algorithm is 100 and was determined empirically. To determine the peak frequency, we calculate the frequency spectrum and determine the prominent peaks of the spectrum with the maximal magnitude. Figure 7.4 shows two sample segments, their energy profiles, and frequency spectrum. The first column is of a snore signal while the second column is of a non-snore signal. They exhibit very different energy profile and frequency spectrum. The snore segment has a more slender energy profile while the noise segment produces a fat and noisy energy profile. In terms of frequency, the snore segment features a few prominent peak frequencies while the noise segment has a smoother spread of frequency.

The energy feature allows us to distinguish snore patterns with different duration and profiles, while the peak frequency helps to weed out non-snore noises. A previous study shows that snore sounds have a frequency component at around 100 Hz [89] while a noise usually exhibits a wide range of frequency values with smooth magnitude change. Figure 7.5 shows five groups of signal segments from

FIGURE 7.5: Hierarchical clustering of piezoelectric sensor signal segments

a subject that can be distinguished by the energy profile. The snore segments in Cluster 2 have a slender energy profile compared to those in Cluster 1, indicating that Cluster 2 has a shorter duration compared to Cluster 1. Cluster 20 and 15 feature two snore events within the 5s duration. The former has a longer snore duration than the latter. The last group (Cluster 6) consists of noise segments with irregular energy profile.

7.2.4.4 Hierarchical Clustering

Instead of the usual clustering methods such as K-mean, we employ a hierarchical-clustering method. The main reason for the use of hierarchical clustering is for future extension. We want to classify segments not only into snore or non-snore class but also into different types of snores. We believe that the hierarchy can provide us with more information about the factors behind the different type of snores. Ultimately, we want to use this extra information to help us study the relationship between snore pattern and one's health condition. For example, Figure 7.5 shows five groups of signal segments with four of them containing snore events. The snore patterns of the four groups are distinct from each other. These snore patterns could be indicators of specific medical issues if a significant number of subjects with the same medical background exhibit the same snore patterns.

The end results of a hierarchical clustering is often presented in the form of dendrogram. A dendrogram groups data into a hierarchy of clusters based on the dissimilarity between the data points. Various distance metrics are available to measure the dissimilarity between audio segments. We use the Euclidean distance, which is given by the L2-norm of the energy distance, d_e , and peak-frequency distance, d_f , between a segment, s_1 , and another segment, s_2 :

$$d(s_1, s_2) = \sqrt{d_e(s_1, s_2)^2 + d_f(s_1, s_2)^2}. \quad (7.2)$$

The energy distance is given by the maximum of the normalized cross-correlation between two energy profiles:

$$d_e(s_1, s_2) = 1 - \max_{-N+1 \leq n \leq N-1} \sum_{m=1}^N \frac{(s_1[m] - \bar{s}_1)(s_2[m+n] - \bar{s}_2)}{\sigma_{s_1} \sigma_{s_2}}, \quad (7.3)$$

where N is the segment length; m is the index of the signals; $\bar{s}_1$, $\bar{s}_2$, $\sigma(s_1)^2$ and $\sigma(s_2)^2$ are their mean and variance of Segment s_1 , and Segment s_2 ; n is the index of the cross-correlation values ranging from $-N + 1$ to $N - 1$. The use of cross-correlation makes the dissimilarity measure between two segments shift-invariant. This is useful since the segmentation step used in the previous step does not guarantee to center the snore signal in a segment. A normalized version of the metric ensures that the measure is magnitude invariant too. This has the implication that snore signals with similar energy profile shape but different magnitude will be grouped together.

The peak-frequency difference is merely the difference in peak frequency between two segments normalized by the Nyquist frequency, $f_s/2$:

$$d_f(s_1, s_2) = \frac{2(f_p(s_1) - f_p(s_2))}{f_s}. \quad (7.4)$$

With the distance metric as shown in Equation 7.2, 7.3, and 7.4, we can construct a distance matrix showing the distances between sample segments. A dendrogram is constructed based on this distance matrix. The top center of Figure 7.5 shows a dendrogram constructed from the 10% five-second segments from a subject. On the left and bottom of the figure are five groups of segments sampled from the leaf nodes (the end of the branch) with a significant number of segments as indicated by the histogram in the top left corner. The segments in each cluster clearly exhibit similar energy profiles, and the difference between clusters are distinctive. Besides looking similar, the distance metrics also indicate the same. The last two plots in the first row of each group display the histogram of the maximum normalized cross-correlation value and maximum peak frequency for each segment, respectively. We can see that the values are concentrated, suggesting that the segments in each group are indeed similar.

7.2.4.5 Human Labeling

In this step, samples from significant clusters are labeled by a human user. The process involves the operator listening to the auralized segments and deciding if the segments contain snore event. In Figure 7.5, segments in red color are those labeled as snore while those in black are non-snore.

Instead of listening to every sample in each cluster, users can choose to listen to a fixed number of samples. In my test, we used 16 samples from each cluster for inspection.

With the labels from the human user and the assumption that segments within a cluster are similar, we can label clusters as snore or non-snore according to the majority of the segments in the clusters. In the example in Figure 7.5, the first four clusters are snore clusters while the last one is not.

In the case where the number of snore and non-snore segments are about the same, further clustering of the original cluster can be carried out until the segments in each sub-clusters are homogeneous, containing mostly either snore or non-snore segments.

7.2.4.6 Classification

With the label of each major clusters established, we can use them as a snore classifier. To achieve this, a representative signal for each cluster is calculated by taking the average of the segments' energy profile and maximum peak frequency. The top left corner of each cluster plots in Figure 7.5 displays the representative energy profile for each group. A new segment is classified into one of these clusters by computing the distance metrics between the segment and the clusters' representative signal using Equation 7.2, 7.3, and 7.4. The new segment is grouped into the cluster that has the smallest dissimilarity metrics with the segment. The segment then takes the label of the cluster.

7.2.5 Experiments

To test the proposed method, we designed a software prototype with a graphical user interface (GUI) to incorporate the work flow detailed in the previous subsection. Next, we used the software to test snore signals from a set of polysomnography (PSG) data from University of Pittsburgh.

7.2.5.1 CVDR Dataset

We performed the test on the Sleep and Cardiovascular Disease Risk (Sleep-CVDR) Study dataset from the Department of Psychiatry of University of Pittsburgh. These data were collected through a grant from the National Institutes of Health (R01 HL104607, M. Hall, PI). The dataset contains 305 set of PSG data generated from 156 subjects, of which 102 are females and 54 are males. Their age ranges from 45 to 85 years old. Of these data, only 238 contains a snore channel recorded using a piezoelectric sensor attached to the neck of the subjects. 44 of them were sampled at 64 Hz, 21 at 256 Hz, and 173 at 1024 Hz. For consistency, we only used those sampled at 1024 Hz.

7.2.5.2 GUI

We created a GUI to facilitate human input in labeling snore. We coded the proposed method in MATLAB and developed the GUI using MATLAB's GUI Development Environment (GUIDE). Figure 7.6 shows the six interfaces in the GUI. The first one has the users input a PSG file in the European Data Format (EDF). The snore channel showing the whole night recording of the piezoelectric signal is then displayed in the interface. Users then click on the "Next" button to trigger the V-Box-based segmentation, uniform sampling, feature extraction and hierarchical clustering. This results in a dendrogram and a histogram showing the number of segments in each leaf node of the dendrogram in the second interface.

The subsequent interface (Figure 7.6 (c)) shows that the algorithm samples segments from each major clusters and display the energy profile of each segment. Users can play each sampled segments and listen to the auralized piezoelectric signal. Next, users decide if a segment contains snore and enter the label through a combo box. If further clustering of the selected cluster is necessary, users can click on the 'Split' button, and an extra sub-clusters split from the original cluster will be added to the major cluster list. This situation happens when neither the snore segment nor the non-snore segment is the dominant majority. After repeatedly doing the labeling for samples from each major clusters, the final classification result is displayed in Figure 7.6 (d) where the snore episodes are marked red.

FIGURE 7.6: GUI of human assisted snore labeling software: (a) PSG data loading; (b) Clustering; (c) Labeling; (d) Classification; (e) Verification; and (f) Results display

7.2.5.3 Verification

To verify the classification results, We went through minute by minute the piezo-electric signal in a recording. We checked if an episode marked in red (snore) are indeed snores. We also checked for any missed snore signal. Figure 7.6 (e) shows the interface that allows us to perform this check by listening to the auralized signal. At the end of the check, the software calculates the number of misses, false snores, true snores, false non-snores, and true non-snores. Please note that this step is only for the purpose of assessing the software and will not be included in the final snore labeling software.

As it is very time-consuming to go through the whole night recording, we only performed this verification test on random 60-second segments of the recording. The total length of verified 60-second segments ranges from 30 minutes to an hour. We made the assumption that the average rate of misses, false snores, true snores, false non-snores, and true non-snores are similar throughout the night. Based on this assumption, we calculate the extrapolated number of missed snores, false snores, and false non-snores for the whole night. Finally, we compute the classification precision, recall, and F1-score. Figure 7.6 (f) displays the classification results of a subject. The metrics include number of true snore, false snore, true non-snore, false non-snore, and miss. They are used to compute the precision, recall, and F1-score. F1-score is computed as follows:

$$F1 = 2 \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \quad (7.5)$$

where precision is defined as:

$$\text{Precision} = \frac{\text{true snore}}{\text{true snore} + \text{false snore}} \quad (7.6)$$

and recall is defined as:

$$\text{Recall} = \frac{\text{true snore}}{\text{true snore} + \text{false non-snore} + \text{miss}}. \quad (7.7)$$

7.2.5.4 Results

We ran the test on 25 subjects from the CVDR dataset. Figure 7.7 shows the number of snore and non-snore segments extracted by the algorithm. Due to space

FIGURE 7.7: Classified piezoelectric signal segments for each subject

limitation, only the statistics of 15 subjects are shown. Nevertheless, assuming the classification is right, the plot indicates there is a good mix of subjects, from light (Subject 28048, 28130, and 47700) to heavy snorers (Subject 28123, 28156, and 47685). It also displays a mix of signal condition from low noise (Subject 28106, 28112, 47685, and 47700) to a significant amount of noise (Subject 28156 and 66111).

Also, note that the smallest number of segments is 581 from Subject 47700. The 10% of the amount gives about 58 segments that are sufficient for clustering. Hence, the use of 10% as the sample amount for clustering is justifiable.

The average extrapolated F1-score is 0.9289 with a standard deviation of 0.0517 as shown in Table 7.1. The result shows that the proposed algorithm can recall most of the snore segments but less precise. This means it tends to misclassify non-snore segments as snore but seldom fails to detect snore segments. We suspect this is due to the bias of labelers, who might try not to miss any snore signal during the labeling process but ended up including non-snore signals. Nevertheless, the F1-Score is on par with other classification results (90.2%, 90%, 90.74%, 98.2%, 95.6%), be it supervised or unsupervised, reported in [87], [89], [88], [91], and [92], respectively. This is despite the use of only two simple features and PSG data set with a low sampling rate.

TABLE 7.1: Average and standard deviation of the extrapolated classification metrics

	Precision	Recall	F1-score
Avg.	0.8940	0.9703	0.9289
Std. Dev.	0.0627	0.0627	0.0517

One of the motivations to develop this algorithm is to reduce the snore labeling time. Hence, we also measure the time taken during the computation, human labeling, and verification process of the last seven subjects. Figure 7.8 shows the processing time breakdown. Computation time includes the time involved in data loading, signal processing, V-Box segmentation, segment clustering, and classification. From the plot, we observe that the cluster labeling takes the bulk of processing time in the semi-automatic snore labeling process (orange bars in Figure 7.8). On average, human users take 21.9 minutes to label clusters.

Another observation is that the verification time (yellow bars in Figure 7.8) is larger than the human labeling time (23.2 minutes on average) except for Subject 28151 and 28198. Given that the total length of the verified 60-second segments for each of these seven subjects is 30 minutes, we can make a crude estimation of the time needed to manually label the full-night recording by extrapolating the verification time with the total sleep time (red curve in Figure 7.8). The estimated full night manual labeling time for each subject is indicated by the purple bars in Figure 7.8. The average time is 5.5 hours. This shows a 93.3% reduction in the labeling time comparing the manual labeling and the semi-automatic labeling process.

Subject 28151 and 28198 required longer human labeling time (34.3 and 58.6 minutes, respectively) because the first run of clustering resulted in clusters with a mix of snore and non-snore segments. Further clustering was conducted to obtain finer clusters with more homogeneous segments. The finer clustering gave improved classification performance from 0.7541 to 0.8269 and from 0.6359 to 0.8489 for Subject 28151 and 28198, respectively.

FIGURE 7.8: Processing time of semi-auto snore labeling

7.3 Conclusion

In this chapter, we tested the idea of condition-specific learning that uses a single source of data for both training and testing to overcome the non-stationary condition. This pre-learning strategy is applied to snore labeling, which is a suitable application as it provides an abundance of data from a single source.

In this application, we have shortened the snore labeling process by combining hierarchical clustering and human input. To verify this method, we developed a GUI and applied it to the CVDR dataset. It achieved a reasonable accuracy of 0.9289 F1-score while significantly shortening the labeling time from hours to minutes.

The use of PSG data and clustering allows us to label snore in large quantities. Furthermore, the use of hierarchical clustering makes it possible to quantify snore beyond just its presence. The algorithm allows the identification of a few distinctive snore patterns, as shown in the experiment. The use of simple features and hierarchy make the classification process transparent and easy to understand for the non-expert.

In conclusion, with the capability to process a large amount of snore data from PSG in a short time and the ability to cluster snore signals into different types using simple features and explainable hierarchy, this algorithm will be a useful software tool for researchers to conduct a large-scale study on the relationship between snore types and health condition.

Chapter 8

Deep Learning: Learning Robust Features

It is not always feasible to use condition-specific learning proposed in Chapter 7. Most of the time, we have to pull data from multiple sources and perform feature engineering to obtain features that are relevant to the target source of variation and are consistent across the sources. A limitation of engineered features is that extensive modeling and experimentation are required to get the best features [28], and these features may not be robust enough to changes in conditions [97]. Deep learning saves us the trouble of feature engineering by incorporating feature learning from raw data. In this chapter, we explore the use of deep learning methods to extract features that are robust to condition change. The work presented in this chapter was first presented in [27].

8.1 Concept

Deep learning has taken the field of machine learning by storm ever since researchers demonstrated the quantum leaps performance that it achieved in image recognition [98] and speech recognition [99]. The success is attributed to its deep structure that enables representation learning directly from raw data [100], unlike conventional methods that require painstaking feature engineering. The multiple layers in the networks transform the input in a way that increases the invariance of the representation but does not reduce its selectivity [101]. In addition, the many hidden

nodes in each layer represent data in a distributed way that allows exponentially many combinations beyond those given by the raw training data, thereby enabling greater generalization [100]. These features of invariance and generalization are attractive to deal with non-stationary conditions.

There are two neural network architectures: feedforward neural networks and recurrent neural networks. Feedforward networks have the inputs flow in a single direction from the input layer towards the output layer of the networks, while recurrent networks redirect the outputs back to the inputs of the networks [100]. Since we are interested in feature learning, we focus on a specialized kind of feedforward networks called convolutional neural networks (CNN) [102] and recurrent neural networks (RNN) [103].

Comparing the two in terms of data inputs, CNN deals with data in an array format, for instance, signals (1D array), images (2D array), and videos (3D array), while RNN focuses on sequential data, such as speech, music, and language [100]. Regarding features learning, CNN learns local connections from array data while RNN looks at long-term dependencies in sequential data, but both architectures employ parameter sharing for efficiency [101]. To discover even longer-term dependencies, memory modules are usually added. One of such kind of RNNs is the long short-term memory (LSTM) networks [104]. We shall make use of these deep learning architectures to extract useful features that are robust to condition change for snore detection in Section 8.2.

8.2 Application: Audio-based Snore Detection

8.2.1 Motivation

As discussed in Chapter 7, sleep quality is important to one's overall well-being. Currently, to correctly diagnose any sleep disorder requires patients to spend a night in the sleep clinic to obtain Polysomnography (PSG) data. Besides being uncomfortable, it is also unknown if the sleep data taken at the sleep clinic is representative as he or she may not sleep well in an unfamiliar setting. Moreover, it is unlikely for an individual to consider visiting a sleep clinic until the condition becomes severe.

For most people, the better solution is to have a device that they can use at home regularly and over a long period. There are many devices on the market that claim to monitor one's sleep health at night. These devices use motion sensors to track users' movements during sleep. The downside of these devices is that users have to wear them while sleeping, which is intrusive and troublesome.

For sleep apnea, there exists a home sleep testing (HST) device, which is a simplified version of PSG. While it saves users a trip to the sleep clinic, the simplified device is still intrusive, impeding users from getting into their natural sleeping condition. Moreover, the collected data again requires a laborious examination by a trained professional.

Given the importance of sleep health, we want to build a non-intrusive, automated, and ambulatory sleep monitoring tool that is complete with snore and breathing anomaly detection for the general public to assess their sleep health. The system is illustrated in Figure 8.1. The home-based and non-intrusive nature of the device provides comfort to the users. It also offers a longitudinal record of the users' sleep health. Combined with the automated detection capability and the additional snore and breathing monitoring, it provides a better picture of one's sleep health. This information, in turn, enables healthcare providers to diagnose sleep-related health issues more accurately and faster, and subsequently provide timely intervention.

Part of the system requires audio-based snore detection. As described in Sub-Section 7.2.2 in the previous chapter, many handcrafted features have been tried by researchers for snore detection. However, there is a lack of analysis of the robustness of these features to subject variation. Given the advantage of deep learning in extracting more invariant features, we explored different deep learning architectures for feature learning to improve the robustness of the classification to subject variation.

8.2.2 Methods

This sub-section describes the data, method, and performance metrics we used to evaluate the robustness of features extracted using different deep learning models: 1D, 2D CNNs, LSTM, and a combination of them for snore classification.

FIGURE 8.1: An non-intrusive (pressure and audio sensing), ambulatory sleep monitoring device offers comfortable and long term monitoring at home

8.2.2.1 Data

We used the same snore dataset (Sleep-CVDR) and labels generated from the semi-automatic snore labeling software in Chapter 7. Of the 28 labeled recordings, only those with an extrapolated F1-score exceeding 0.9 were used. This data selection step is to reduce the complication caused by wrong snore labeling. This step resulted in 19 recordings with 81207 snore and non-snore five-second segments.

Depending on the type of deep learning methods, the data used can be in its raw 1D signal form or transformed into a 160×160 image (2D signal) in the form of a Mel-spectrogram, as shown in Figure 8.2. Mel-spectrogram of a signal is the log power spectrum of the signal mapped on a nonlinear Mel-scale of frequency. Mel-scale is used because it approximates the pitch scale perceived by human listeners. The 1D waveform shows the signals only in the temporal domain, but the 2D Mel-spectrogram has an extra frequency axis to display the signal in the spectral domain. The discrete cosine transform of the Mel-spectrogram along the frequency axis yields the Mel-frequency-cepstrum coefficients (MFCC), which are often used as the handcrafted features for audio signals.

8.2.2.2 Feature Learning Models

Figure 8.3 shows the different deep learning architectures used for snore detection: fully connected neural networks (FCN), 1D CNN, 2D CNN, LSTM, VGG, 1D

FIGURE 8.2: Five-second snore segment as a 1D waveform (Top) and its corresponding 2D Mel-spectrogram. The resonant formant frequencies of the snore are clearly shown as the layered texture in Mel-spectrogram but hard to see in the 1D waveform.

CNN-LSTM, and 2D CNN-LSTM. Conventional machine learning algorithms such as the support vector machine (SVM) was also included for comparison. These methods were chosen methodologically to compare the effect of feature learning, classifier types, convolution, and recurrent neural networks, as illustrated in the cuboid.

On the top face of the cuboid, we compare different deep learning techniques such as CNN and LSTM. They are also combined modularly to form more complex neural networks: 1D CNN-LSTM and 2D CNN-LSTM. The addition of LSTM aimed to capture long term patterns in the temporal space of the snore signals.

Besides differing in network structures, the deep learning models also work on different inputs. 1D CNN works on 1D snore signals, as illustrated by a blue waveform and labeled as an audio signal in Figure 8.3. It extracts temporal features from the audio signal. On the other hand, 2D CNN works on 2D snore images (Mel-spectrograms). Since the Mel-spectrograms contain both temporal and spectral information, we expect the 2D CNN to perform well in snore classification. In addition, we also include a deep convolutional network, VGG19 [105], on the premise

FIGURE 8.3: Illustration of the model ablation study for snore classification: Each face of the cuboid represent different comparison matrices, namely, classifier and features (front), deep learning architectures (top), different 2D CNN features (side). The five arrows around the cuboid show the axes of comparison.

that an increased network depth improves accuracy by making the features learned more invariant, as discussed before. Given the large computational resources to train the deep network from scratch, we use the model pre-trained on ImageNet data [5] instead.

Besides the deep learning techniques, we also include other traditional methods as a control. On the front face, we compare the effect of handcrafted features, such as MFCC, with features learned from different CNNs. The idea behind this comparison is that we want to know if feature learning through deep learning really has an edge over handcrafted features. Lastly, a comparison between a support vector machine (SVM) and a fully connected neural networks (FCN) was performed too. The motivation is to determine if any change in performance is attributed to better feature extraction or better classification capability.

The detailed network structures of the different models are summarized in Table 8.1. The hyper-parameters used to train the different models are summarized in Table 8.2.

TABLE 8.1: Neural network structure of deep learning models used for feature learning. Notation: Conv(number of kernel, kernel size, stride, padding); MaxPool(kernel size, stride, padding); Linear(Number of output); BatchNorm( ); Dropout(dropout probability); LSTM(input size, hidden state size)

Model	Layers		
	Input	Hidden	Output
1D CNN	5121	Conv1D(16, 3, 1, 2) with ReLU MaxPool1D(4, 4, 0) Conv1D(16, 4, 1, 2) with ReLU MaxPool1D(4, 4, 0) Conv1D(16, 5, 1, 2) with ReLU MaxPool1D(4, 4, 0) Conv1D(16, 5, 1, 2) with ReLU MaxPool1D(4, 4, 0) with Dropout(0.6)	Linear(2)
2D CNN	160x160	Conv2D(16, 5x5, (1,1), (2,2)) with ReLU BatchNorm2D( ) MaxPool2D(2, 2, 0) Conv2D(16, 5x5, (1,1), (2,2)) with ReLU BatchNorm2D( ) MaxPool2D(2, 2, 0) Conv2D(16, 5x5, (1,1), (2,2)) with ReLU BatchNorm2D( ) MaxPool2D(2, 2, 0) with Dropout(0.8)	Linear(2)
1D CNN-LSTM	5121	Conv1D(10, 4, 1, 2) with ReLU MaxPool1D(4, 4, 0) Conv1D(10, 4, 1, 2) with ReLU MaxPool1D(2, 2, 0) with Dropout(0.5) LSTM(100, 200)	Linear(2)
2D CNN-LSTM	160x160	Conv2D(16, 5x5, (1,1), (2,2)) with ReLU BatchNorm2D( ) MaxPool2D(2, 2, 0) Conv2D(16, 5x5, (1,1), (2,2)) with ReLU BatchNorm2D( ) MaxPool2D(2, 2, 0) Conv2D(16, 5x5, (1,1), (2,2)) with ReLU BatchNorm2D( ) MaxPool2D(2, 2, 0) with Dropout(0.4) LSTM(320, 200)	Linear(2)
VGG	160x160	Pre-Trained from TensorFlow framework [106]	-
FCN	3240	Linear(128) with ReLU and Dropout(0.3) Linear(128) with ReLU and Dropout(0.3) Linear(128) with ReLU and Dropout(0.3)	Linear(2)

TABLE 8.2: Hyper-parameters used to train each snore classification model.

Model	Learning rate	Epoch	Batch size	Dropout	C
1D CNN	0.001	10	200	0.6	-
2D CNN	0.01	4	200	0.8	-
1D CNN-LSTM	0.001	4	200	0.5	-
2D CNN-LSTM	0.001	4	50	0.4	-
VGG19-SVM	-	-	-	-	1
MFCC-SVM	-	-	-	-	1
MFCC-FCN	0.001	10	32	0.3	-

8.2.2.3 Leave-One-Out Model Evaluation

To test the robustness of the features to subject variation, we used an evaluation scheme that mimics the leave-one-out cross-validation. Given N (in this case, 16) available subjects, we trained each of the models using $N - 1$ subjects and tested it on the remaining subject. This procedure was repeated N times until all the subjects are tested. In this way, we can test if the features learned from other $N - 1$ subjects work well on the leftover subject. By doing this over all the subjects, we obtained statistics of the classification performance that tells us the robustness of the features across changing subjects (conditions).

8.2.2.4 Performance Metrics

To determine the accuracy of the classification, An F1-score was computed for each held-out subject:

$$F1 = 2 \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \quad (8.1)$$

where the precision is defined as:

$$\text{Precision} = \frac{\text{true snore}}{\text{true snore} + \text{false snore}} \quad (8.2)$$

and the recall is defined as:

$$\text{Recall} = \frac{\text{true snore}}{\text{true snore} + \text{false non-snore}}. \quad (8.3)$$

These metrics were used because the dataset has imbalanced classes (see Figure 7.7).

Different from other classification problems, we do not concern only the accuracy but also the robustness. We use mean and median F1-scores to measure the snore classification accuracy and statistical dispersion metrics, such as standard deviations (SD) and an inter-quartile ranges (IQR), to measure of robustness to subject variation. The magnitude of dispersion indicates how variable the classification accuracy due to subject change. Hence, a robust classification should have a small dispersion.

The use of medians and IQRs besides means and standard deviations is to avoid a small number of extreme data points skewing the metrics. Medians and IQRs are not affected by extreme data points. In an ideal situation, a feature is said to improve the classification accuracy if both the mean and median F1-scores increase. Similarly, for improvement in robustness, both standard deviations and IQRs should decrease.

8.2.3 Results

Figure 8.4 shows the precision, recall, and F1-score statistics from the leave-one-out experiment. The more detailed numerical values of the F1-score statistics are shown in Table 8.3.

From both the box plot in Figure 8.4 and Table 8.3, we can see that VGG-SVM is the clear winner. It not only produced the highest accuracy with a mean F1-score of 0.7929 and a median of 0.8012 but also has the most robust snore classification with the smallest dispersion at 0.1355 (standard deviation) and 0.1955 (IQR). These values are reflected in the box-plot. The good accuracy is indicated by the highest ‘x’, which represents the average score, and horizontal marking in the box, which represents the median score. Its box length, which measures the IQR, being the shortest, tells us that it is the most robust.

The best performance of VGG-SVM is followed closely by 1D CNN-LSTM with a mean and median value that are 0.0058 and 0.0154 smaller than VGG-SVM, respectively. Its standard deviation and IQR are, respectively, only 0.0104 and 0.0138 less than VGG-SVM.

Surprisingly, at a mean and median F1-score of 0.7395 and 0.7149, 2D CNN has the worst performance in terms of accuracy, as illustrated by the box with the

FIGURE 8.4: Leave-one-out model evaluation results of seven snore classification models: The three groups of box plots show the statistics of the precision, recall, and f1-score from 16 subjects. The mean values are marked by 'x' in the plot, while the median by the horizontal lines within the boxes. Both ends of the boxes represent the first and third quartile of the metrics. The whiskers show the minimum and maximum with the maximal length at 1.5 times the inter-quartile range (IQR). Any point outside the whiskers is considered an outlier.

TABLE 8.3: Ablation study to see the incremental effect of different features and classifiers (NN: Neural Networks, SVM: Support Vector Machine): The values in brackets show the differences compared to the VGG SVM, which have the best mean, median, standard deviation, and inter-quartile range (IQR) of the F1-score.

Method	Feature	Classifier	F1-Score			
			Mean	Median	Std. Dev.	IQR
1D CNN	Temporal	NN	0.7395(-0.0533)	0.7149(-0.0863)	0.1674(0.0319)	0.2590(0.0635)
1D NN-LSTM	Long-term Temporal	NN	0.7871(-0.0058)	0.7858(-0.0154)	0.1459(0.0104)	0.2093(0.0138)
2D CNN	Temporal & Spectral	NN	0.7026(-0.0902)	0.6987(-0.1025)	0.1707(0.0352)	0.2815(0.0860)
2D CNN-LSTM	Long-term Temporal & Spectral	NN	0.7378(-0.0550)	0.7583(-0.0429)	0.1727(0.0372)	0.2333(0.0378)
MFCC-FCN	Spectral	NN	0.7282(-0.0647)	0.7287(-0.0725)	0.1579(0.0224)	0.2179(0.0224)
MFCC-SVM	Spectral	SVM	0.7447(-0.0481)	0.7610(-0.0402)	0.1704(0.0349)	0.2163(0.0209)
VGG-SVM	Long-term Temporal & Spectral	SVM	0.7929	0.8012	0.1355	0.1955

lowest position in Figure 8.4. With the longest box plot, its robustness is also bad. It scored the largest IQR at 0.0860 and the second largest standard deviation at 0.0352.

Other models have similar performance in terms of snore classification accuracy and robustness. In an ascending order of accuracy measured by mean F1-score, the list starts with MFCC-FCN, followed by 2D CNN-LSTM, 1D CNN, and MFCC-SVM. For robustness measured by the IQR of F1-score, the order is slightly different: 1D CNN, 2D CNN-LSTM, MFCC-FCN, and MFCC-SVM.

From Figure 8.4, we can also observe that the recall values are higher on average compared to the precision values. In other words, the models seldom miss snore segments but have a greater tendency to misclassify non-snore segments as snores. In terms of dispersion measure, all the precision values have longer boxes, pointing to more varied performance across different subjects. The observation suggests that the features captured by those models are better at identifying snores correctly than distinguishing non-snore segments from snore segments. The features learned to distinguish non-snore segments in one subject does not work so well on the other subject. The lower and disperse precision could be an indicator that there are more sources of variation in non-snore segments across subjects.

8.2.4 Discussion

In this sub-section, we would like to discuss the effect of different features captured by the seven different models (listed in Column 3 in Table 8.3). Specifically, we will compare the features in the following dimension: temporal versus spectral, handcrafted versus learned, short-term versus long-term, and shallow versus deep. We will also discuss the effect of classifier choice.

8.2.4.1 Effect of the Classifier Choice

In the ablation study, we compared two different classifiers: a support vector machine (SVM) and a fully connected neural networks (FCN) using the same handcrafted features, MFCC. The SVM had a better mean and median F1-score (0.7447 and 0.7610, respectively) compared with FCN (0.7282 and 0.7287, respectively).

In terms of robustness, both methods are similar. SVM produced a standard deviation of 0.1704 and an IQR of 0.2163, while FCN yielded 0.1579 for the standard deviation and 0.2179 for the IQR. From these observations, we see that neural networks as a classifier alone do not produce superior classification accuracy and robustness compared to SVM. In fact, SVM is better in terms of accuracy. Therefore, we can exclude the classifier effect in our subsequent analysis. In other words, any difference in classification performance is attributed to the feature difference rather than the choice of the classifier.

8.2.4.2 Handcrafted vs. Learned Features

Comparing handcrafted and learned features, we observe that not all learned features perform better than handcrafted features in accuracy and robustness. Only VGG-SVM and 1D CNN-LSTM achieved better accuracy and robustness than MFCC-FCN and MFCC-SVM.

For better comparison, we look at VGG-SVM and MFCC-SVM. Both use the same classifier, but VGG-SVM performs feature learning through a deep CNN, and MFCC-SVM uses handcrafted MFCC. We can see that VGG-SVM performs much better than MFCC-SVM in terms of accuracy (+0.0481 in mean and +0.0402 in median) and robustness (-0.0349 in standard deviation and -0.0209 in IQR).

We can also compare MFCC-FCN with 1D CNN-LSTM since both use neural networks as the classifier. We observe a similar improvement between MFCC-FCN and 1D CNN-LSTM.

From these observations, we can conclude that with the right network for feature learning, learned features outperform handcrafted features in both the accuracy and robustness.

8.2.4.3 Temporal vs. Spectral Features

First, we compare the difference between spectral and temporal features. Moving from 1D CNN, which learns only spectral features, to 2D CNN, which learns both temporal and spectral features encoded in the Mel-spectrogram, the mean F1-score drops by 0.0369. The decrease in accuracy is unexpected since we expect additional

information from the spectral domain should help to distinguish snore segments from non-snore segments better. Judging from the reduced precision, the addition of spectral features seems to confuse 2D CNN in distinguishing non-snore from snore segments.

For MFCC-FCN and MFCC-SVM, the spectral feature (MFCC) also did not give them an extra edge in terms of accuracy and robustness compared to 1D CNN. Though the mean F1-score improves from 0.7395 to 0.7447 for MFCC-SVM, it decreases to 0.7282 for MFCC-FCN. A similar trend is observed for robustness; the standard deviation of F1-score improves to 0.1579 for MFCC-FCN but worsens to 0.1074 for MFCC-SVM. Note that both MFCC-SVM and MFCC-FCN do register improvement in terms of median and IQR. Hence, the drop in the mean score is caused by the data points near the extreme ends (first and last quartiles).

In a nutshell, the inclusion of spectral features improves neither accuracy nor robustness. Instead, the performance drops for 2D CNN. The reason behind the drop could be due to the translation invariance nature of convolution. While this is useful along the temporal axis, it is harmful along the frequency axis.

8.2.4.4 Short-term vs. Long-term Features

Adding LSTM cells, which extract long-term dependency information, to 1D CNN to create 1D CNN-LSTM results in an increment in the mean F1-score by 0.0475, an increment in the median F1-score by 0.0709, a decrease in the standard deviation by 0.0215, and a decrease in the IQR by 0.0497. For 2D CNN, there are improvements in accuracy and robustness, although with a smaller magnitude. The improvements are 0.0352 for the mean F1-score, 0.0596 for the median F1-score, and 0.0482 for the IQR. In conclusion, long-term features help to improve both the accuracy and robustness of the snore classification.

8.2.4.5 Shallow vs. Deep Networks

We should also assess the impact of using deep networks. To do that, we compare VGG-SVM with similar methods. First, we compare VGG-SVM with 2D CNN. Both methods extract spectral features through 2D convolutional layers. The differences are that VGG-SVM has a deeper architecture, and an SVM classifier is

used. By using a deep network, we see a big improvement in performance in terms of accuracy and robustness: +0.0902 for the mean F1-score, +0.1025 for the median F1-score, -0.0352 for the standard deviation of F1-score, and -0.0860 for the IQR.

Next, we can make a comparison with MFCC-SVM, since they both use SVM as the classifier, and run the input features through layers of networks. In addition, MFCC is derived from the Mel-spectrogram. Though the difference could be partly due to the handcrafted-learned factor, we see that VGG-SVM significantly improves the mean and median, from 0.7447 to 0.7929 and 0.7610 to 0.8012, respectively; as well as the standard deviation and IQR, from 0.1704 to 0.1355 and 0.2163 to 0.1955, respectively.

Though none of the comparisons are perfect in quantifying exactly the effect of deep networks, it is safe to say that features extracted from the deep networks contribute to better accuracy and robustness. In fact, it has the best performance in accuracy and robustness among the seven models.

The good performance can be explained by the fact that the VGG-SVM model is pre-trained on the large-scale ImageNet data (14 million images from 1000 classes of images). This large dataset provides better generalization capability and thus better accuracy and robustness for snore classification. Besides, it could also be the result of the very deep architecture of VGG-Net. Recent research in deep learning-based sequence modeling shows that very deep convolutions architectures can capture long-term dependencies too and do so more effectively than recurrent neural networks [107]. This study also explains why adding LSTM cells improve both accuracy and robustness.

8.3 Conclusion

Audio-based snore detection is an integral part of the proposed non-intrusive and home-based sleep monitoring device. The challenge in achieving high accuracy in snore detection is the lack of robustness of various extracted features across different subjects. Another concern is that traditional feature engineering requires tedious experimentation, even with in-depth domain knowledge, to extract good features.

In addition, most of the research studies were done based on overall accuracy. Few of them study the robustness of the features to subject variation.

In this chapter, we attempted to bridge the gaps by studying the feature robustness to changing subjects by using a leave-one-out model evaluation test. Besides, we also explored different deep learning techniques for robust feature learning. The results show that, for snore classification, very deep networks and recurrent networks that extract features with long term dependencies yields not only the most accurate but also the most robust classification results. The features learned through these networks beat handcrafted features, such as MFCC.

In conclusion, we can improve supervised learning performance under changing conditions (across different subjects in this chapter) by using deep learning techniques to learn more robust features. Since deep learning requires a large amount of data for training, this pre-learning technique is applicable if a large dataset is available.

Chapter 9

Artificial Data: Generating Your Own Data

In Chapters 7 and 8, the two pre-learning strategies work on pre-existing datasets. In this chapter, we generate data through computer simulation. This ability allows us to address non-stationary conditions by controlling the data generation process. We will employ this technique to teach a welding robot climbing down from a tall wall using reinforcement learning. The conceptual design of the welding robot presented in this chapter has been published in [108].

9.1 Motivation

9.1.1 Challenges in Shipbuilding Automation

Modern large ships are constructed by assembling prefabricated double-hull blocks. Figure 9.1 shows a mid-ship bottom segment of a bulk carrier and the two constituting double-hull block types: double-bottom and bilge blocks. During the construction, welders perform welding in confined spaces which subject them to occupational hazards like toxic welding dust, high temperatures, and postural problems. The confined space also limits the working time, preventing the industry from achieving its maximum efficiency. Automating the welding process will not only help the welders but also increase the efficiency of the industry.

FIGURE 9.1: Shipbuilding by assembling double-hull blocks.

The adoption of automation in the shipbuilding industry is still low because the majority of existing welding robots are immobile and it is difficult to move fixed-base robotic manipulators into double-hull blocks due to their many internal reinforcement structures, such as longitudinal stiffeners (abbreviated as longis) and carlings, as shown in Figure 9.1. Thus, there is a need for mobile robotic manipulators, which can overcome these structures to reach the welding sites in the blocks autonomously.

Automating the double-hull welding process presents a few unique challenges. Firstly, a robot needs to be able to access the blocks through small and elevated holes. Secondly, while inside the blocks, it needs to navigate through protruding internal structures to reach the welding sites. Lastly, the robot has to perform welding in a constrained space at the welding sites.

9.1.2 Double-Hull Blocks Welding Robot

9.1.2.1 Conceptual Design

To address the challenges in automating the welding inside double-hull blocks, we proposed a mobility solution using a chained track drive as shown in Figure 9.2 (a). It is made up of three links joined by two powered revolute joints, and each link is equipped with a tracked drive. To increase the stability and maneuverability, the center link features two tracks. This feature allows the center section to carry a manipulator and perform on-the-spot turns using a differential drive.

The three-link design enables the robot to exhibit multiple configurations, as shown in Figure 9.2 (b), to shift between a bridging and conforming mechanism. The conforming mechanism allows the robot to fold up the two links (see Configuration 1 and 8 in Figure 9.2 (b)) to adapt to the terrain and maneuver in tight spaces, such as those between longis (Figure 9.2 (c)) and tapered corners inside a bilge block (Figure 9.2 (d)). On the other hand, the bridging mechanism is realized by using the two long links to lift the central link to overcome obstacles. Figure 9.2 (e) and (f) show the robot using this mechanism to bridge across manholes (Configuration 3 in Figure 9.2 (b)) and longis (Configuration 2 in Figure 9.2 (b)), respectively.

Further analysis was conducted in [108] to determine the feasible dimension of the robot. Besides, we also empirically verified the design using a 1/3 scale prototype.

9.1.2.2 Full-Size Prototype

With the conceptual design verified, we proceeded to construct a full-size double-hull blocks welding robot. Figure 9.3 (a) and (b) show the dimension of the robot and the final assembled robot with major components labeled, respectively. In addition to the three tracked links, we also fitted the robot with an industrial manipulator (Denso VS060) on the central link. Sensors are needed for successful mobility and welding manipulation. Figure 9.3 (b) shows a LIDAR sensor (Velodyne-Puck 16) and a 2D laser scanner (Micro-Epsilon ScanControl LLT2750-100) attached to the manipulator.

(a) Three tracked links

(b) Possible configurations

(c) Fitting between longis

(d) Accessing a tapered corner

(e) Climbing through a manhole

(f) Bridging across longis

FIGURE 9.2: Conceptual design

(a) CAD prototype

(b) Physical prototype

FIGURE 9.3: Full size prototype

9.1.2.3 Software

To operate all the different systems of the welding robot at the same time and have them working together in coordination require software that can perform multi-processing and is equipped with inter-process communication. Robotic Operating System (ROS) [109] provides an excellent software architecture for us to develop such software.

Software development for robots requires testing on physical robots. However, testing on physical hardware is expensive because of the potential damage to the robot due to erroneous software. The solution to this problem is to use a computer simulator for testing during the early stage of software development and perform the final test on physical robots. We adopted this strategy for software development and constructed a simulator of the welding robot. To this end, we used GAZEBO [110]. It is an open-source robot simulation software. It comes with physic engines that can simulate the dynamic interaction between the simulated robot and its simulated environment. It also provides plugins, modular pieces of programs, to simulate sensors, actuators, and motion controllers.

9.1.2.4 Robot Test

We want to verify whether the assembled robot can perform its intended mobility and welding function. Hence, we sent the robot to the shipyard of our project sponsor in Japan and conducted an experiment there. In this experiment, we want to verify if the robot can climb through a manhole, position itself on top of longis near the welding site, and perform automated welding at the welding site. In order not to interrupt the production of the shipyard, a cutout of the double bottom block including just one welding site was prepared.

Figure 9.4 shows the snapshots of the robot performing different maneuvers for mobility and welding. The first row of images shows how the robot accessed the block through a manhole using its long links to lift itself and tracks to propel itself forward. In the second row of the images, the robot, with its arms folded, performed an on-the-spot rotation in the tight space between the longis. It then used its long tracked links to climb up onto longis. With the robot well-positioned, the manipulator was deployed (see the third row of Figure 9.4) and performed

FIGURE 9.4: Testing the mobility and welding function of the full-size welding robot on a cutout block.

an autonomous scanning to register the welding site and determine the welding trajectory. Finally, it performed the welding.

9.1.3 Bilge Block Climbing: How to Land from High Walls

In the previous sub-section, we have verified that the full-size double-hull welding robot is able to access a double bottom block, but we did not test it on another type of double-hull block, the bilge blocks (see Figure 9.1). Hence, we conducted another test on a cutout section of the bilge block that features a tall wall. Figure 9.5 shows the snapshots of the robot climbing the cutout section of the bilge block.

In the first figure, we can see that the wall is taller than the arm. To reach the wall, the robot had to combine the length of both its body and arm (second figure of Figure 9.5). By clamping the arms and body tightly on the wall, making three contact points with the wall, the robot used its body and arm track to climb up the wall in the third figure. In the last figure, we see that the robot is sitting on top of the wall with both arms off the ground. We stopped the test at this point as we did not know how to land the robot safely.

Note that throughout the test, the manipulator was not fitted on the robot. Furthermore, we used steel wire ropes to secure the robot to an overhead crane. This is because we lacked the confidence to land the robot without toppling and damaging the robot.

FIGURE 9.5: Full-size prototype scaling a high wall.

9.1.4 Simulator: Safer Environment

To explore a climbing down solution safely, we can make use of the simulator constructed in Sub-Section 9.1.2. Figure 9.6 shows our attempt to manually controlling the simulator to climb over the tall wall. The first three figures show the climbing-up process and the last two depict the climbing-down process. We observe that the robot topple very often during landing (the last figure in Figure 9.6) when we tried the motion plan proposed in [108], where all three links maintain contacts with the wall while climbing down (see the fourth figure in Figure 9.6). We wonder if there exists a safer solution.

Using the simulator, we can fit the manipulator on the robot without the risk of damaging the manipulator. During the simulation, we also notice that the robot is especially prone to toppling when the manipulator is extended forward. This observation prompts another question: can the manipulator play a role in making the landing safer?

9.1.5 Reinforcement Learning: Learning to Land Safely

To explore a safer way of landing and the use of a manipulator for that purpose, we propose the use of reinforcement learning. In a reinforcement learning setting, an agent learns how to react (action) to what it senses (observation/state) from the environment through trial-and-error. The state-action mapping (policy) is shaped by a reward scheme. Generally, we grant positive rewards for desired behaviors and negative rewards for undesired behaviors. This scheme encourages the mapping to choose actions that lead to positive rewards and avoid negative rewards. In essence, reinforcement learning algorithms learn the mapping between state and action from data consists of action, state, and reward that are generated from trials.

Reinforcement learning [111] is a very useful tool for roboticists as it provides a simple framework for robots to acquired hard-to-engineer behaviors through trial-and-error [112]. However, in practice, it is not as simple for robots to learn in the real environment due to uncontrollable external variables [112], concerns about safe exploration [113], and slow learning speed. As a result, most researchers resort to computer simulation.

FIGURE 9.6: Welding robot simulator scaling a high wall.

Acquiring a safe landing policy through reinforcement learning and simulation poses another challenge. Simulation speeds up the learning process, but the success in simulation rarely transfers to the real world. Simulation bias [112] is the term used when the policy learned from simulation fails in the real-world setting. This failure can be seen as another example of changing conditions. The main reason for the failure is that the simulation does not model the real-world setting accurately. The uncertainty in the modeling, especially the physical interaction (contact and friction dynamics), makes it difficult to transfer a control model learned from a simulation to the real world.

9.2 Artificial Data Generation

In the previous section, we learned that reinforcement learning methods establish state-action mapping from state-action-reward data. The reliance on data means that reinforcement learning methods suffer from non-stationary conditions too, as exemplified by the difficulty of transferring a learned policy from simulation to the real world. Fortunately, reinforcement learning algorithms generate their own data through trials-and-errors. Coupled with the use of simulation that gives us control of the environment, non-stationary conditions can be addressed by choosing the environment setting wisely to generate data.

In Chapter 2, we mentioned that the ideal data cover every possible scenario that will be encountered during the deployment phase. However, it is generally difficult to gather such kind of data. One of the few possible ways is through computer simulation.

Since we have full control over the simulation environment, a naive solution is to run many simulations to cover all possible settings in the hope that one of them matches the real-world setting. The data generated this way will help a learning agent to generalize to the real world. However, the amount of data required is huge.

Instead of generating all possible situations, the better way is to use a Monte Carlo approach: doing a random sampling from a pool of settings. One recent success in applying this method is OpenAI's dexterous in-hand manipulation learning [114],

where the controller learned the generic policy through many random friction settings. In this way, the controller overcomes the uncertainty in friction between fingers and manipulated objects.

Knowing the possible range of conditions encountered in the deployment stage helps us to choose the pool of settings for random sampling. We can design the simulation environment such that it covers a wide enough number of scenarios. In this way, an agent can learn a generic model that applies to our target deployment condition. We shall test this idea to teach the mobile welding robot to climb down safely from a tall wall.

9.3 Experiment

In this section, we employ a deep reinforcement learning technique called Deep-Q-Network (DQN) [115, 116] to teach the welding robot simulator to climb down safely from a tall wall. In addition, we test the idea of selective artificial data generation proposed in the previous section.

9.3.1 Setup

For this experiment, we use a combination of software frameworks. We simulate the tall wall section of the bilge block and the welding robot using GAZEBO. The robot simulator is controlled using ROS architecture. Finally, OpenAI's GYM [117] and Stable Baseline [118] are used to implement reinforcement learning.

9.3.1.1 Simulation Environment

The simulation environment consists of a cutout section of the bilge block and the welding robot simulator. Figure 9.7 shows the cutout section. Note that the actual cutout section of the bilge block only has a longi but two longis are used in the cutout for structural integrity.

Since this is our first attempt of using reinforcement learning to train the robot and deep reinforcement learning is notoriously hard to train [119], we scale down

FIGURE 9.7: A cutout section from a bilge block was used as a simulation environment for the welding robot simulator.

the problem scope by limiting the robot motion to a 2D plane. We achieve this by setting up two invisible walls parallel to the longis as shown in Figure 9.7. GAZEBO achieves such an invisible feature by only setting the collision geometry but not the visual geometry.

9.3.1.2 Reinforcement Learning Setup

To use reinforcement learning techniques for control, we first model the safe-landing control problem as a sequential decision problem. In this setting, at each time step (0.3 s), the robot has to decide which of the six available actions: (i) no action, (ii) lifting front arm, (iii) lowering front arm, (iv) lifting rear arm, (v) lowering rear arm, (vi) shifting manipulator forward, and (vii) shifting manipulator backward, to take. Figure 9.8 (a) shows the joint actuation to realize the above actions. The actions are discrete and the step sizes are 0.9 degrees for both the front and rear arm joint and 9 degrees for the second joint of the manipulator. Note that we do not actuate the arm and body tracks to keep the action space small. The track actions can be added in the future as an extension.

To complete the control process, feedback is required. The action decision is made based on the current observable status of the robot. We choose the following observable variables: (i) front arm joint angle, (ii) rear arm joint angle, (iii) second joint angle of the manipulator, (iv) pitch angle of the robot's body, (v) pitch rate of the robot's body, (iv) roll angle of the robot's body, and (v) roll rate of the robot's

FIGURE 9.8: Actions and states in our reinforcement learning formulation: (a) shows the available joint action for action selection; (a) and (b) form the states of the robot.

body to fully describe the current status of the robot. These variables are shown in Figure 9.8. For 2D planar motion, the last two variables are not necessary, but we keep them for future expansion.

Modeling the control process as a Markovian decision process (MDP), the transition from the current state, $\mathbf{s}$, to a new state, $\mathbf{s}'$, after an action, a , has the following probability:

$$P(\mathbf{s}'|\mathbf{s}, a). \quad (9.1)$$

The difference between reinforcement learning and MDP is that the transition model, $P(\mathbf{s}'|\mathbf{s}, a)$, is not given but rather learned from data. For this, we use function approximation methods such as neural networks to represent the transition model.

For each state transition, a reward, $R(\mathbf{s})$ is awarded. Note that we use the convention that an award, $R(\mathbf{s})$, is received when an agent reaches a new state, $\mathbf{s}$. In our case, most of the transition produces a zero reward, except the one that ends with the robot safely landed. For this success, a +5 reward is awarded. This reward scheme was determined through trial-and-error.

To select a new state to transit to, and hence the action needed for the transition, we look at the possible reward in the new state. As the reward is sparse (in our case,

only the successful end state has a reward), we use a value function to value each state. The value function, $U(\mathbf{s})$, is the sum of reward at the current state, $R(\mathbf{s})$, and discounted expected reward from future states. It is given by the Bellman equation:

$$U(\mathbf{s}) = R(\mathbf{s}) + \gamma \max_{a \in A} \sum_{\mathbf{s}'} P(\mathbf{s}' | \mathbf{s}, a) U(\mathbf{s}'), \quad (9.2)$$

where γ is the discount factor, and A is the action space. With the value function determined, the optimal policy, $\pi(\mathbf{s})$, which is defined as the action, a , taken at each state, $\mathbf{s}$ to maximize the expected future reward:

$$\pi^*(\mathbf{s}) = \arg \max_{a \in A} \sum_{\mathbf{s}'} P(\mathbf{s}' | \mathbf{s}, a) U(\mathbf{s}'). \quad (9.3)$$

It is cumbersome to split the action selection process into two stages, where we first determine the value function at each state before selecting an action. Hence, we use action-value function (Q-value) which merges both value function and action selection. Replacing the value function with the action-value function gives the action-value version of the Bellman equation:

$$Q(\mathbf{s}, a) = R(\mathbf{s}) + \gamma \max_{a' \in A} \sum_{\mathbf{s}'} P(\mathbf{s}' | \mathbf{s}, a) Q(\mathbf{s}', a'). \quad (9.4)$$

With the action-value function, the largest action-value directly indicates the policy:

$$\pi^*(\mathbf{s}) = \arg \max_{a \in A} Q(\mathbf{s}, a). \quad (9.5)$$

Deep-Q-Network approximates the action-value function using a neural network (Q-network) [115]. Instead of learning the transition model, the network directly learns from samples of state-action-reward $(\mathbf{s}, a, R(\mathbf{s}, a))$ generated during reinforcement learning. Besides the use of neural networks, another important difference between DQN and traditional Q-learning is that memory replay, which collects past state-action-reward, is used as the sampling pool for the network update. In this experiment, we use the DQN function from Stable-Baseline [118] as the DQN code has been verified and benchmarked using standard GYM environments such as CartPole and Atari.

As learning is done iteratively, the simulation needs to be reset. We set the maximum number of steps for each episode of the simulation to be 200. The episode ends

early when the robot topples from the wall or the manipulator touches the ground. The last ending condition is necessary because, during our experimentation, the robot learned to use the manipulator as support to land from the wall.

During reset, we need to set the initial joint angles for all the joints shown in Figure 9.8 (a) and also the initial position of the robot on the wall. The setting of the initial condition range is important because it affects the variety of data generated during reinforcement learning. To show this, we prepared two sets of initial joint angle range, as shown in Table 9.1. The first set has a wider range, while the second has a narrow range. The initial conditions were randomly selected from a uniform distribution of these ranges. Another consideration in setting the range of initial joint angles is making sure the robot does not topple from the beginning.

TABLE 9.1: Range of initial condition for welding robot climbing-down simulation.

Set	θ_1 (deg)		θ_2 (deg)		θ_3 (deg)	
	Min	Max	Min	Max	Min	Max
Wide	50	90	50	90	-90	90
Narrow	80	90	80	90	-60	60

We ran the reinforcement learning using the DQN parameter setting shown in Table 9.2. For each set of the initial condition range, we ran the training for 200,000 steps and chose the best model encountered during the first 100,000 steps. The selection is made by evaluating the model at every 10,000 steps of learning using 50 fixed initial conditions generated from each of the two initial condition ranges. We also evaluated the model trained with the wide initial condition range using the fixed conditions from the narrow range, and vice versa.

9.3.2 Results and Discussion

Figure 9.9 shows the evaluation result of the reinforcement learning model for both the wide and narrow initial condition range. All four combinations achieve the best performance at Step 80,000, except the Wide $\times$ Narrow combination at Step 90,000. The combination of Wide $\times$ Narrow and Narrow $\times$ Narrow has the best average reward at 4.5. Narrow $\times$ Wide combination only managed to reach 3.5 while Wide $\times$ Wide achieved 3.9.

TABLE 9.2: DQN parameters.

Parameter	Value
learning rate	0.0005
neural network	64×64
batch size	256
learning starts after	10000 step
buffer size	10000
prioritized replay α	0.6
prioritized replay β_0	0.4
prioritized replay ϵ	1×10^{-6}
exploration fraction	0.6
exploration final ϵ	0.05

(a) Wide (training) $\times$ Wide (evaluation)

(b) Wide (training) $\times$ Narrow (evaluation)

(c) Narrow (training) $\times$ Wide (evaluation)

(d) Narrow (training) $\times$ Narrow (evaluation)

FIGURE 9.9: Evaluation results of reinforcement learning models trained with wide and narrow initial condition range using fixed initial condition with wide and narrow range.

For the model trained with the wide initial condition range, the result of evaluation at both the wide and narrow range shows comparable average rewards (3.9 versus 4.5). The increase in performance is because of the narrow initial condition range offering a more stable starting condition.

On the other hand, the performance of the model trained with the narrow initial condition range drops from 4.5 to 3.5 when evaluate at the wide initial condition range. This observation suggests that the shift of condition reduces the model performance. A non-stationary condition has occurred.

Comparing the two models, the one trained with the wide initial condition range produces a more robust model. The performance did not suffer when it was evaluated at a different initial condition range. It improved instead due to the more stable starting conditions. The use of a wider range of initial condition allows the simulation to generate data with more variety. This variety, in turn, helps to train a more robust model.

Due to its more robust performance, we chose the control model trained with the wide initial condition range for further evaluation. Using the model at Step 80,000, the robot has two modes of safe-landing, as shown in the first two rows of Figure 9.10. The first success mode uses the front arm to tilt the robot forward. When the robot is pitching forward, the control model moves the manipulator backward to shift the center of gravity backward and slow down the pitch rate. In the second mode, the robot makes use of the rear arm as well to balance the front arm.

Interestingly, when the robot starts from a difficult position where the robot is already pitched forward (see the third row of Figure 9.10) and is merely supported by the front arm, the controller is able to recover. It first uses a clamping motion to push the robot back on top of the wall. During the clamping, the manipulator moves to the back to shift the center of gravity backward. This motion also helps to put the robot back on top of the wall. Then, once the robot is on top of the wall, it continues with the safe landing procedures shown in the first row.

Besides toppling (the fourth row of Figure 9.10), the controller fails when it runs out of time. This failure happens when it tries to recover from a bad starting position.

FIGURE 9.10: Snapshots of evaluating landing control model learned from reinforcement learning.

9.4 Conclusions

In this chapter, we have successfully used a reinforcement learning technique to explore sequences of action that allow the double-hull welding robot to climb down safely from a tall wall. The trained controller exhibited two safe-landing modes that used both arms and a manipulator for balancing during the controlled landing. The best landing success rate is 90%.

We also showed that reinforcement learning suffers from non-stationary conditions too. The performance of the controller trained with a narrow initial condition range dropped when it was evaluated with a wider initial condition range. This gave us the idea of artificial data generation. Under this pre-learning strategy, we control the variety of the data generated through the initial conditions of simulations.

The results in this chapter are preliminary as we have not tried other more advanced reinforcement learning techniques such as Asynchronous Actor-Critic (A2C) [120], Hindsight Experience Replay (HER) [121], Generative Adversarial Imitation Learning (GAIL) [122], Proximal Policy Optimization (PPO) [123], and Trust Region Policy Optimization (TRPO) [124]. These methods have shown better performance than DQN. Also, the parameters used may not be optimal yet. Future works should include further tuning of the DQN parameters and experimenting with other methods. Besides, the idea of artificial data generation shows encouraging preliminary results, as reported in this chapter. We want to extend this idea further to transfer the control model trained in simulations to the real-world physical robot.

Chapter 10

Conclusions

10.1 Summary of Contribution

This thesis focuses on ways to improve machine learning methods when non-stationary conditions, such as covariate shift and concept shift, occur. A non-stationary condition happens when a deployment condition deviates from a training condition, causing a drop in the performance of the trained prediction model.

Importance weighted learning (IWL) is the conventional method to address covariate shifts. It uses a ratio of test feature distribution to training feature distribution to weight the training data during learning. This ratio implicitly selects data more relevant to the test condition to train the prediction model and consequently improves the prediction performance under covariate shift. However, by applying IWL in a toy inverse kinematics problem, we reveal that it cannot adapt to frequent shifts in condition as it is both a batch learning and batch prediction method. Moreover, it can only deal with covariate shifts but not concept shifts.

In search of different methods that can address the shortfalls of IWL, we propose looking at the three different stages of supervised learning: pre-learning, in-learning, and post-learning. Such taxonomy gives us not only a larger repertoire of tools to tackle the problem, but also a unified view on the strategy we can employ to address non-stationary conditions. Under this new perspective, IWL is classified as an in-learning method. In-learning methods modify the training process to adapt to a change in condition while post-learning methods modify the prediction

process instead. Pre-learning methods reduce the adverse effect of non-stationary conditions through data preparation and robust feature learning.

The taxonomy also serves as a selection guide. When limited test data are available during the training phase, in-learning methods can be deployed. When time urgency and frequent shifts in condition are the problems, we choose post-learning methods. Finally, if data are abundant, the pre-learning strategy is the choice.

Besides IWL, we developed another in-learning method that makes use of the idea of transfer learning. After pre-training a neural network with a generic training dataset, re-training the network with an additional small amount of test data improves the prediction performance in vocal emotion recognition. For example, an amount of test data equivalent to half of the training data improves the prediction accuracy by 10%.

The need for re-training when a condition changes makes in-learning methods not suitable for problems that require faster response to deal with frequent shifts in condition. Hence, we look at post-learning methods that modify the prediction in response to a condition change. We developed a post-learning method using the concept of local learning. It uses distance metrics in feature space to weight and combine predictions from multiple local prediction models. Applying this method to the same inverse kinematics problem shows that it not only addresses covariate shifts equally well as IWL but do so in an online fashion. The advantage allows it to deal with frequent covariate shift. In our vocal emotion recognition test, our local learning method also outperformed IWL in terms of accuracy. In the arousal classification of the EmoDB dataset, the local learning method achieved 93.2% accuracy on average compared to 88.8% for IWL under covariate-shift conditions.

Extending from the local learning method, we developed a feature-augmentation technique that converts a difficult concept-shift problem into a simpler covariate-shift problem. The method expands the feature space to include recently matched feature-label data. As this expansion is done in an online learning setting, it is suitable for problems that require a timely response and exhibit frequent changes. One such problem is water shooting under changing winds or on a moving vehicle. In our experiment, we showed that the control model developed using the feature-augmentation method outperforms both conventional PID controllers and human operators.

Though both IWL and local learning methods can address covariate shifts, we need to identify first whether a covariate shift occurs. Applying those methods to non-covariate-shift problems may not improve the prediction results. At worst, it may reduce the performance. Using feature augmentation allows local learning to handle generic shifts, but extending the method to a higher dimension increases quickly the dimension. In search of an alternative, we look at pre-learning methods. They are promising because they look directly at dataset and features, which are the most effective way to improve machine learning results.

One way of using the pre-learning strategy is condition-specific learning. Assuming data abundance, it uses a single source of data for both training and test to prevent the complication caused by non-stationary conditions. We applied this strategy to automate snore labeling. The resulting snore labeling software not only achieves good accuracy (0.93 F1-score) but also cuts the labeling time from hours to minutes.

Besides dealing with data, we also looked at feature learning as a way of doing pre-learning compensation. To this end, we explored the use of deep learning to learn features that are robust to condition changes. Through an ablation study, we found that a very deep network and recurrent network, which extract features with long term dependencies, yield not only the most accurate but also the most robust snore classification results. This study shows that we can use deep learning techniques to improve supervised learning performance under non-stationary conditions.

In the final part of the thesis, we detail the design and construction of a double-hull welding robot. The robot has a three-tracked-link design that allows it to achieve mobility inside a double-hull block with tight spaces and protruding internal structures while carrying a heavy manipulator for welding. The design is a combination of conforming and bridging mechanisms, informed by our survey of obstacle scaling in mobile robots. We tested the hardware design for mobility and welding using a cutout section of a double bottom block. The test results show that the robot is capable of performing its intended mobility and welding function. Extrapolating the time measured in the test to cover the full double bottom block shows that it has the potential capability of seven to ten welders.

To extend the usage of the robot to bilge blocks, another type of double-hull blocks, the robot needs to be capable of climbing down safely from a tall wall. We proposed the use of reinforcement learning techniques to construct a safe-landing controller

from data. Using a deep-Q-network (DQN), we developed a controller that achieves a 90% safe landing success rate. Besides the learning-based controller, we also showed that reinforcement learning could suffer from non-stationary conditions. Fortunately, the combination of reinforcement learning and computer simulation affords us the use of artificial data generation as another pre-learning strategy to address non-stationary conditions. In our experiment, we showed that by increasing the variety of starting conditions during the training phase of reinforcement learning, the resulting controller is more robust to a change in condition.

In summary, by looking at a broader picture of supervised learning, we are no longer limited to only in-learning methods such as IWL to combat non-stationary conditions. We have access to other tools like post-learning and pre-learning methods. The choice of method depends on data availability, time urgency, and type of condition change. These methods have shown their effectiveness in our applications. We used condition-specific learning to construct a semi-automatic snore labeling software that cut the labeling time from hours to minutes. The performance of vocal emotion recognition, which is vital for social robots to connect to humans more empathically, was improved by transfer learning and local learning. Combining local learning and feature augmentation, we devised a learning-based controller that outperforms both PID controllers and human operators. Lastly, we employed artificial data generation through reinforcement learning and computer simulation to help a double-hull welding robot to learn a more robust way of climbing down safely from a tall wall.

10.2 Future Work

Of all three strategies to combat non-stationary conditions, pre-learning is the most promising. This is because this class of methods deals with data and features that have the most effect on machine learning results. In this thesis, we have used pre-learning methods such as learning robust features through deep learning. Though we have seen improved overall robustness of snore classification by using deep learning features, we still observe poor performance in some subjects (see Chapter 8), suggesting that the features learned are not truly robust yet. Unsupervised feature learning may be a better way. In supervised deep learning, labels are used to guide feature learning. The label-guided feature learning selects features that improve

the model selectivity (the ability to discriminate one class from another) in the training condition but does not guarantee all features selected are robust (the ability to generalize to different conditions). Doing away labels might help us to learn more robust features. Unsupervised learning is also attractive because it works on unlabeled data, which we can gather more easily. It is also worth noting most of the learning in nature is unsupervised. Hence, a natural extension to robust features learning is using unsupervised deep learning networks.

Unsupervised deep learning networks have been used to extract features automatically. Some examples of deep networks trained unsupervisedly include stacked autoencoders [125], convolutional deep believe networks (DBN) [126], and generative adversarial network (GAN) [127]. Autoencoders capture salient features by optimizing a fixed criterion such as sparsity in sparse autoencoders [128], reconstruction error between the noisy input and actual input in denoising autoencoders [129], penalizing derivatives in contractive autoencoders [130], etc. The downside is that the fixed criterion used in those autoencoders defines the feature saliency [100]. The features considered salient by the criterion might not be relevant. In contrast, GAN learns how to determine saliency [100]. It allows us to provide conditions to guide the feature extraction process. Hence, as future work, we would like to use GAN to extract robust features.

Lastly, the reinforcement learning model tested in Chapter 9 showed interesting results, but we have not exhausted all available methods to improve the results further. Future work includes testing other more advanced reinforcement learning techniques such as Asynchronous Actor-Critic (A2C) [120], Hindsight Experience Replay (HER) [121], Generative Adversarial Imitation Learning (GAIL) [122], Proximal Policy Optimization (PPO) [123], and Trust Region Policy Optimization (TRPO) [124]. Besides, we would like to extend the idea of artificial data generation to transfer control models trained in simulation to real-world robots.

List of Author's Publications

Journal Articles

- **Chun Fan Goh** , Akshay Hinduja, Divish Ajmani, Robin Song, Lei Zhang, and Kenji Shimada, “Designing a Mobility Solution for Fully Autonomous Double Hull Block Welding,” *ASME Journal of Mechanism and Robotics*.

Conference Proceedings

- **Chun Fan Goh**, Gerald Gim Lee Seet, and Kenji Shimada, “A model-based learning controller with predictor augmentation for non-stationary conditions and time delay in water shooting,” in *13th IEEE Conference on Automation Science and Engineering (CASE)*, 2017.
- **Chun Fan Goh**, Laura B. Samuelsson, Martica H. Hall, Gerald Gim Lee Seet, and Kenji Shimada, “Semi-Automatic Snore Detection in Polysomnography based on Hierarchical Clustering,” in *14th IEEE Conference on Automation Science and Engineering (CASE)*, 2018.

Poster Session

- Akhil Chaturvedi, **Chun Fan Goh**, Rahul Shailesh Raina, and Kenji Shimada, “Comparing Deep Feature Representations to Improve Robustness to Subject Variation in Snore Detection,” in *SLEEP*, 2019.

Bibliography

- [1] Dimitrios Ververidis and Constantine Kotropoulos. Emotional speech recognition: Resources, features, and methods. *Speech Communication*, 48(9):1162–1181, 2006. ISSN 0167-6393. doi: 10.1016/j.specom.2006.04.003. URL <http://www.sciencedirect.com/science/article/pii/S0167639306000422>. xxvii, 36, 37, 38, 41
- [2] Roddy Cowie and Randolph R Cornelius. Describing the emotional states that are expressed in speech. *Speech communication*, 40(1):5–32, 2003. xxvii, 38
- [3] Kangbeom Cheon, Jaehoon Kim, Moussa Hamadache, and Dongik Lee. On replacing pid controller with deep learning controller for dc motor system. *Journal of Automation and Control Engineering Vol.*, 3(6), 2015. 1
- [4] S. Schaal and C. G. Atkeson. Learning control in robotics. *IEEE Robotics Automation Magazine*, 17(2):20–29, June 2010. ISSN 1070-9932. doi: 10.1109/MRA.2010.936957. 1, 24, 75
- [5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 1, 124
- [6] Vladimir Naumovich Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999. 2
- [7] Stationary process. https://en.wikipedia.org/wiki/Stationary_process. Accessed: August 15, 2018. 3
- [8] João Gama, Indrè Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014. 3, 4, 72
- [9] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009. 3, 48
- [10] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 45(1):521–530, 2012. 3

- [11] Indre Zliobaite. Learning under concept drift: An overview. Technical report, Vilnius University, 2009. 4, 6, 71
- [12] Ali Hassan, Robert Damper, and Mahesan Niranjan. On acoustic emotion recognition: compensating for covariate shift. *Audio, Speech, and Language Processing, IEEE Transactions on*, 21(7):1458–1468, 2013. 4, 48
- [13] M. Sugiyama and M. Kawanabe. *Machine Learning in Non-stationary Environments: Introduction to Covariate Shift Adaptation*. Adaptive computation and machine learning. MIT Press, 2012. ISBN 9780262017091. URL <https://books.google.com/books?id=Akd-8xFKac4C>. 4, 5, 21, 22, 23, 25, 48, 61
- [14] Makoto Yamada, Masashi Sugiyama, and Tomoko Matsui. Covariate shift adaptation for semi-supervised speaker identification. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 1661–1664. IEEE, 2009. 4
- [15] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. Covariate shift adaptation by importance weighted cross validation. *The Journal of Machine Learning Research*, 8:985–1005, 2007. 4, 5
- [16] Jiayuan Huang, Arthur Gretton, Karsten M Borgwardt, Bernhard Schölkopf, and Alex J Smola. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608, 2006. 4, 5
- [17] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000. 4, 5
- [18] Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative learning under covariate shift. *The Journal of Machine Learning Research*, 10: 2137–2155, 2009. 5, 22
- [19] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul V Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in neural information processing systems*, pages 1433–1440, 2008. 5, 22
- [20] Takafumi Kanamori, Shohei Hido, and Masashi Sugiyama. A least-squares approach to direct importance estimation. *The Journal of Machine Learning Research*, 10:1391–1445, 2009. 5
- [21] Amos Storkey. chapter When training and test sets are different: characterizing learning transfer, pages 3–28. Cambridge, MA: MIT Press, 2009. 5, 49

- [22] H Mouss, D Mouss, N Mouss, and L Sefouhi. Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. In *Control Conference, 2004. 5th Asian*, volume 2, pages 815–818. IEEE, 2004. [6](#)
- [23] J Zico Kolter and Marcus A Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research*, 8:2755–2790, 2007. [6](#)
- [24] George Forman. Tackling concept drift by temporal inductive transfer. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 252–259. ACM, 2006. [6](#)
- [25] Chun Fan Goh, Gerald Gim Lee Seet, and Kenji Shimada. A model-based learning controller with predictor augmentation for non-stationary conditions and time delay in water shooting. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1110–1117. IEEE, 2017. [9](#), [71](#)
- [26] Chun Fan Goh, Laura B Samuclsson, Martica H Hall, Gerald Gim Lee Seet, and Kenji Shimada. Semi-automatic snore detection in polysomnography based on hierarchical clustering. In *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, pages 1116–1122. IEEE, 2018. [9](#), [97](#)
- [27] Akhil Chaturvedi, Chun Fan Goh, Rahul Shailesh Raina, and Kenji Shimada. 0449 comparing deep feature representations to improve robustness to subject variation in snore detection [poster]. *Sleep*, 42(Supplement_1):A180–A181, 2019. doi: 10.1093/sleep/zsz067.448. [9](#), [119](#)
- [28] Pedro M Domingos. A few useful things to know about machine learning. *Commun. acm*, 55(10):78–87, 2012. [15](#), [98](#), [119](#)
- [29] Carlos Oscar Sánchez Sorzano, Javier Vargas, and A Pascual Montano. A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*, 2014. [15](#)
- [30] John P Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *The Journal of Machine Learning Research*, 16(1):2859–2900, 2015. [15](#)
- [31] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014. [15](#)
- [32] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017. [15](#)

- [33] Gretton AarthurDepartment Empirical Inference; Smola AJsmola; Huang JhuangDepartment Empirical Inference; Schmittfull MmslDepartment Empirical Inference; Borgwardt KMkarstenDepartment Empirical Inference; Schölkopf BbsDepartment Empirical Inference. *Covariate Shift by Kernel Mean Matching*, pages 131–160. MIT Press, Cambridge, MA, USA, 2009. ISBN 0-262-25510-3. [22](#), [53](#)
- [34] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997. [24](#)
- [35] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive Processing*, 12(4):319–340, 2011. ISSN 1612-4782. [24](#), [75](#)
- [36] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):9, 2016. [31](#)
- [37] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010. [32](#)
- [38] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International Conference on Artificial Neural Networks*, pages 270–279. Springer, 2018. [32](#)
- [39] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014. [32](#)
- [40] Ronald C Kessler, Wai Tat Chiu, Olga Demler, and Ellen E Walters. Prevalence, severity, and comorbidity of 12-month dsm-iv disorders in the national comorbidity survey replication. *Archives of general psychiatry*, 62(6):617–627, 2005. [33](#)
- [41] MD Jeffrey A Lieberman. Psychiatric care shortage: What the future holds, 2010. URL <http://www.medscape.com/viewarticle/727435>. [33](#)
- [42] A. Tapus, M. J. Mataric, and B. Scasselati. Socially assistive robotics [grand challenges of robotics]. *Robotics & Automation Magazine, IEEE*, 14(1):35–42, 2007. ISSN 1070-9932. [34](#)
- [43] Nicu Sebe, Ira Cohen, Theo Gevers, and Thomas S Huang. Emotion recognition based on joint visual and audio cues. In *18th International Conference on Pattern Recognition (ICPR’06)*, volume 1, pages 1136–1139. IEEE, 2006. [34](#)
- [44] Roberto Barra, Juan Manuel Montero, J Macias-Guarasa, LF D’haro, Rubén San-Segundo, and Ricardo Córdoba. Prosodic and segmental rubrics in emotion identification. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages I–I. IEEE, 2006. [34](#)

- [45] Anja Austermann, Natascha Esau, Lisa Kleinjohann, and Bernd Kleinjohann. Prosody based emotion recognition for mexi. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1138–1144. IEEE, 2005. 34
- [46] N Fragopanagos and John G Taylor. Emotion recognition in human–computer interaction. *Neural Networks*, 18(4):389–405, 2005.
- [47] Sanaul Haq, Philip JB Jackson, and James Edge. Audio-visual feature selection and reduction for emotion classification. In *Proc. Int. Conf. on Auditory-Visual Speech Processing (AVSP08), Tangalooma, Australia*, 2008. 34
- [48] Zeng Zhihong, M. Pantic, G. I. Roisman, and T. S. Huang. A survey of affect recognition methods: Audio, visual, and spontaneous expressions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(1):39–58, 2009. ISSN 0162-8828. doi: 10.1109/tpami.2008.52. 34
- [49] Paul Ekman and Wallace V Friesen. Constants across cultures in the face and emotion. *Journal of personality and social psychology*, 17(2):124, 1971. 35
- [50] S. Haq and P.J.B. Jackson. *Machine Audition: Principles, Algorithms and Systems*, chapter Multimodal Emotion Recognition, pages 398–423. IGI Global, Hershey PA, Aug. 2010. 35
- [51] Felix Burkhardt, Astrid Paeschke, Miriam Rolfes, Walter F Sendlmeier, and Benjamin Weiss. A database of german emotional speech. In *Ninth European Conference on Speech Communication and Technology*, 2005. 35
- [52] M Brookes. “voicebox: Speech processing tool for matlab, 2014. URL <http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html>. 36
- [53] Sira Gonzalez and Mike Brookes. A pitch estimation filter robust to high levels of noise (pefac). In *Signal Processing Conference, 2011 19th European*, pages 451–455. IEEE, 2011. 36
- [54] Sherif M Yacoub, Steven J Simske, Xiaofan Lin, and John Burns. Recognition of emotions in interactive voice response systems. In *INTERSPEECH*, 2003. 41
- [55] Aaron Steinfeld, Terrence Fong, David Kaber, Michael Lewis, Jean Scholtz, Alan Schultz, and Michael Goodrich. Common metrics for human-robot interaction. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 33–40. ACM, 2006. 48
- [56] Alexey Tsymbal. The problem of concept drift: definitions and related work. Technical report, Department of Computer Science, Trinity College, Dublin, Ireland, 2004. 48

- [57] Léon Bottou and Vladimir Vapnik. Local learning algorithms. *Neural computation*, 4(6):888–900, 1992. [49](#), [79](#)
- [58] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning. *Artif. Intell. Rev.*, 11(1-5):11–73, Feb 1997. ISSN 0269-2821. doi: 10.1023/A:1006559212014. [49](#), [79](#)
- [59] David G Lowe. Similarity metric learning for a variable-kernel classifier. *Neural computation*, 7(1):72–85, 1995. [50](#)
- [60] Stefan Schaal and Christopher G Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998. [50](#), [56](#), [75](#), [80](#)
- [61] Peter John Millington. *Associative reinforcement learning for optimal control*. PhD thesis, Massachusetts Institute of Technology, 1991. [50](#)
- [62] H Ritter and K Schulten. Topology conserving mappings for learning motor tasks. *Neural networks for computing*, 151:376–380, 1986. [50](#)
- [63] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991. [50](#)
- [64] Marcos Salganicoff. Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artificial Intelligence Review*, 11(1-5):133–155, 1997. [53](#)
- [65] Masashi Sugiyama and Amos J Storkey. Mixture regression for covariate shift. In *Advances in Neural Information Processing Systems*, pages 1337–1344, 2006. [53](#)
- [66] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990. [56](#)
- [67] Wikipedia contributors. Kolmogorov–smirnov test — wikipedia, the free encyclopedia, 2019. [Online; accessed 15-November-2019]. [61](#)
- [68] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013. [75](#)
- [69] A Emili. *Introduction to Time-Delay Systems*. Springer International Publishing, 2014. [75](#)
- [70] Julio E Normey-Rico and Eduardo F Camacho. Dead-time compensators: A survey. *Control engineering practice*, 16(4):407–428, 2008. [75](#)
- [71] Jean-Pierre Richard. Time-delay systems: An overview of some recent advances and open problems. *Automatica*, 39(10):1667–1694, 2003. [75](#)

- [72] Leonid Mirkin and Zalman J Palmor. Control issues in systems with loop delays. In *Handbook of Networked and Embedded Control Systems*, pages 627–648. Springer, 2005. 75
- [73] Min Wu, Yong He, and Jin-Hua She. *Stability Analysis and Robust Control of Time-Delay Systems*. Springer, 2010. 75
- [74] Pooria Joulani, András György, and Csaba Szepesvári. Online learning under delayed feedback. *arXiv preprint arXiv:1306.0686*, 2013. 75
- [75] Chris Mesterharm. On-line learning with delayed label feedback. In *Algorithmic Learning Theory*, pages 399–413. Springer, 2005. 75
- [76] Sethu Vijayakumar, Aaron D’souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17(12):2602–2634, 2005. 80
- [77] Naomi Altman and Martin Krzywinski. Points of significance: Sources of variation, 2014. 97
- [78] Leon Bottou. Feature engineering. <https://www.cs.princeton.edu/courses/archive/spring10/cos424/slides/18-feat.pdf>. Accessed: 2019-11-13. 98
- [79] Frank B Hu, Walter C Willett, JoAnn E Manson, Graham A Colditz, Eric B Rimm, Frank E Speizer, Charles H Hennekens, and Meir J Stampfer. Snoring and risk of cardiovascular disease in women. *Journal of the American College of Cardiology*, 35(2):308 – 313, 2000. ISSN 0735-1097. doi: [https://doi.org/10.1016/S0735-1097\(99\)00540-9](https://doi.org/10.1016/S0735-1097(99)00540-9). URL <http://www.sciencedirect.com/science/article/pii/S0735109799005409>. 98
- [80] R D’Alessandro, C Magelli, G Gamberini, S Bacchelli, E Cristina, B Maggiani, and E Lugaesi. Snoring every night as a risk factor for myocardial infarction: a case-control study. *BMJ*, 300(6739):1557–1558, 1990. ISSN 0959-8138. doi: 10.1136/bmj.300.6739.1557-a. URL <http://www.bmj.com/content/300/6739/1557.2>. 98
- [81] Markku Partinen and Heikki Palomaki. Snoring and cerebral infarction. *The Lancet*, 326(8468):1325 – 1326, 1985. ISSN 0140-6736. doi: [https://doi.org/10.1016/S0140-6736\(85\)92625-X](https://doi.org/10.1016/S0140-6736(85)92625-X). URL <http://www.sciencedirect.com/science/article/pii/S014067368592625X>. Originally published as Volume 2, Issue 8468. 99
- [82] H Palomäki. Snoring and the risk of ischemic brain infarction. *Stroke*, 22(8):1021–1025, 1991. ISSN 0039-2499. doi: 10.1161/01.STR.22.8.1021. URL <http://stroke.ahajournals.org/content/22/8/1021>.
- [83] J.-P. Neau, J.-C. Meurice, J. Paquereau, J.-J. Chavagnat, P. Ingrand, and R. Gil. Habitual snoring as a risk factor for brain infarction. *Acta Neurologica Scandinavica*, 92(1):63–68, 1995. ISSN 1600-0404. doi: 10.1111/j.1600-0404.

- 1995.tb00468.x. URL <http://dx.doi.org/10.1111/j.1600-0404.1995.tb00468.x>. 99
- [84] JA Hedner, I Wilcox, and CE Sullivan. Speculations on the interaction between vascular disease and obstructive sleep apnea. *Lung biology in health and disease*, 71:823–846, 1994. 99
- [85] RA Cathcart, DW Hamilton, MJ Drinnan, GJ Gibson, and JA Wilson. Night-to-night variation in snoring sound severity: one night studies are not reliable. *Clinical Otolaryngology*, 35(3):198–203, 2010. 99
- [86] Laura B Samuelsson, Anusha A Rangarajan, Kenji Shimada, Robert T Krafty, Daniel J Buysse, Patrick J Strollo, Howard M Kravitz, Huiyong Zheng, and Martica H Hall. Support vector machines for automated snoring detection: proof-of-concept. *Sleep and Breathing*, 21(1):119–133, 2017. 99
- [87] M Cavusoglu, M Kamasak, O Erogul, T Ciloglu, Y Serinagaoglu, and T Akcam. An efficient method for snore/nonsnore classification of sleep sounds. *Physiological measurement*, 28(8):841, 2007. 99, 114
- [88] Asela S Karunajeewa, Udantha R Abeyratne, and Craig Hukins. Silence–breathing–snore classification from snore-related sounds. *Physiological Measurement*, 29(2):227, 2008. 99, 114
- [89] Azadeh Yadollahi and Zahra Moussavi. Automatic breath and snore sounds classification from tracheal and ambient sounds recordings. *Medical Engineering and Physics*, 32(9):985–990, 2010. 99, 106, 114
- [90] Kun Qian, Christoph Janott, Zixing Zhang, Clemens Heiser, and Björn Schuller. Wavelet features for classification of vote snore sounds. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 221–225. IEEE, 2016. 99
- [91] Ali Azarbarzin and Zahra Moussavi. Unsupervised classification of respiratory sound signal into snore/no-snore classes. In *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pages 3666–3669. IEEE, 2010. 100, 106, 114
- [92] Ganjun Ma, Biao Xue, Hong Hong, Xiaohua Zhu, and Zhiyong Wang. Unsupervised snore detection from respiratory sound signals. In *Digital Signal Processing (DSP), 2015 IEEE International Conference on*, pages 417–421. IEEE, 2015. 100, 106, 114
- [93] Martica H Hall, Karen A Matthews, Howard M Kravitz, Ellen B Gold, Daniel J Buysse, Joyce T Bromberger, Jane F Owens, and MaryFran Sowers. Race and financial strain are independent correlates of sleep in midlife women: the swan sleep study. *Sleep*, 32(1):73–82, 2009. 100

- [94] Dennis A Dean, Ary L Goldberger, Remo Mueller, Matthew Kim, Michael Rueschman, Daniel Mobley, Satya S Sahoo, Catherine P Jayapandian, Licong Cui, Michael G Morrical, et al. Scaling up scientific discovery in sleep medicine: the national sleep research resource. *Sleep*, 39(5):1151–1164, 2016. [100](#)
- [95] Susan Redline, Peter V Tishler, Tor D Tosteson, John Williamson, Kenneth Kump, Ilene Browner, Veronica Ferrette, and Patrick Krejci. The familial aggregation of obstructive sleep apnea. *American journal of respiratory and critical care medicine*, 151(3):682–687, 1995. [100](#)
- [96] Ewaryst Rafajlowicz, Mirosław Pawlak, and Ansgar Steland. Nonparametric sequential change-point detection by a vertically trimmed box method. *IEEE Transactions on Information Theory*, 56(7):3621–3634, 2010. [102](#)
- [97] Michalis Papakostas, Evaggelos Spyrou, Theodoros Giannakopoulos, Giorgos Siantikos, Dimitrios Sgouropoulos, Phivos Mylonas, and Fillia Makedon. Deep visual attributes vs. hand-crafted audio features on multidomain speech emotion recognition. *Computation*, 5(2):26, 2017. [119](#)
- [98] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. [119](#)
- [99] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012. [119](#)
- [100] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*, volume 1. MIT press Cambridge, 2016. [119](#), [120](#), [159](#)
- [101] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015. [119](#), [120](#)
- [102] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990. [120](#)
- [103] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning, 2015. [120](#)
- [104] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. [120](#)
- [105] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. [123](#)

- [106] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. [125](#)
- [107] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018. [133](#)
- [108] Chun Fan Goh, Akshay Hinduja, Divish Ajmani, Robin Song, Lei Zhang, and Kenji Shimada. Designing a mobility solution for fully autonomous welding of double-hull blocks. *Journal of Mechanisms and Robotics*, 11(4):045002, 2019. [135](#), [137](#), [143](#)
- [109] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009. [140](#)
- [110] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004. [140](#)
- [111] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press, 1998. [143](#)
- [112] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11): 1238–1274, 2013. doi: 10.1177/0278364913495721. URL <https://doi.org/10.1177/0278364913495721>. [143](#), [145](#)
- [113] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. *CoRR*, abs/1205.4810, 2012. URL <http://arxiv.org/abs/1205.4810>. [143](#)
- [114] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177v2*, 2018. [145](#)
- [115] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. [146](#), [149](#)
- [116] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. [146](#)

- [117] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. 146
- [118] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. <https://github.com/hill-a/stable-baselines>, 2018. 146, 149
- [119] Alex Irpan. Deep reinforcement learning doesn’t work yet. <https://www.alexirpan.com/2018/02/14/rl-hard.html>, 2018. 146
- [120] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016. 154, 159
- [121] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2017. 154, 159
- [122] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning, 2016. 154, 159
- [123] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. 154, 159
- [124] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2015. 154, 159
- [125] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007. 159
- [126] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009. 159
- [127] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 159
- [128] Y-lan Boureau, Yann L Cun, et al. Sparse feature learning for deep belief networks. In *Advances in neural information processing systems*, pages 1185–1192, 2008. 159

- [129] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008. [159](#)
- [130] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pages 833–840. Omnipress, 2011. [159](#)