

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Inference of User Intent in Adaptive Input Interfaces /

Permalink

<https://escholarship.org/uc/item/1t97m9rw>

Author

Cheamanunkul, Sunsern

Publication Date

2014

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

Inference of User Intent in Adaptive Input Interfaces

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Sunsern Cheamanunkul

Committee in charge:

Professor Yoav Freund, Chair
Professor Kamalika Chaudhuri
Professor John Hildebrand
Professor Bhaskar Rao
Professor Lawrence Saul

2014

Copyright
Sunsern Cheamanunkul, 2014
All rights reserved.

The dissertation of Sunsern Cheamanunkul is approved, and it is acceptable in quality and form for publication on micro-film and electronically:

Chair

University of California, San Diego

2014

DEDICATION

To my parents, Somnouk and Nongluk, who have always been my
greatest supporters.

TABLE OF CONTENTS

Signature Page		iii
Dedication		iv
Table of Contents		v
List of Figures		vii
List of Tables		ix
Acknowledgements		x
Vita		xi
Abstract of the Dissertation		xii
Chapter 1	Introduction	1
	1.1 The Automatic Cameraman (TAC)	3
	1.2 The uRight System	4
	1.3 Thesis Organization	6
Chapter 2	User Engagement Identification on the Automatic Cameraman (TAC)	8
	2.1 System Architecture	8
	2.2 User Engagement Identification	10
	2.3 Face Localization	11
	2.3.1 Margin-based Active Training Methodology	13
	2.3.2 Skin-color Detector	16
	2.3.3 Face Detector	18
	2.3.4 Face Tracker	19
	2.4 Face Recognition	21
Chapter 3	Non-convex Boosting and Random Label Noise	32
	3.1 Background and Related Work	32
	3.2 BrownBoost and RobustBoost	36
	3.3 Adaptive- ϵ Heuristic	46
	3.4 Experiments	49
	3.5 Conclusions	53
Chapter 4	uRight: Co-adaptive Handwriting Recognition System	58
	4.1 Co-adaptive Handwriting Recognition System	60
	4.2 System Architecture and Implementation	63
	4.3 Adaptive Recognition Algorithms	66

Chapter 5	Co-adaptation in Handwriting Recognition	78
	5.1 Handwriting Recognition as a Communication Channel . . .	79
	5.2 Experiment	82
	5.3 Results and Discussion	83
	5.4 Conclusions	85
Chapter 6	Improved kNN Rule for Small Training Sets	94
	6.1 Introduction	94
	6.2 Background	97
	6.3 Minimizing KL-Divergence Rule	97
	6.4 Experiments	101
	6.5 Discussion	104
	6.6 Conclusions	106
Chapter 7	Conclusions and Recommendations	113
Bibliography	116

LIST OF FIGURES

Figure 1.1: Text entry rate and training time of various methods.	5
Figure 2.1: Frontal view of TAC	9
Figure 2.2: Layout of the area in which TAC is set up	10
Figure 2.3: User interaction with TAC.	12
Figure 2.4: Face localization unit on TAC.	13
Figure 2.5: Example of boosting score distribution	14
Figure 2.6: Annotated images for skin detection	17
Figure 2.7: Results of the skin-color detector after the first iteration of active training	23
Figure 2.8: Results of the skin-color detector after several iterations of training	24
Figure 2.9: Results of the skin-color detector with and without majority voting .	24
Figure 2.10: On-screen button on TAC	25
Figure 2.11: Face detection features	26
Figure 2.12: Face detection mask	27
Figure 2.13: Score distributions of the face detector over multiple iterations of training	28
Figure 2.14: Examples of face images categorized by their classification scores .	29
Figure 2.15: An output from the face tracker	29
Figure 2.16: Result of the face recognition	30
Figure 2.17: Error rate of the face recognition unit on TAC over multiple sessions.	31
Figure 3.1: Potential functions of different boosting algorithms	36
Figure 3.2: A single iteration of the chip game	38
Figure 3.3: RobustBoost algorithm.	47
Figure 3.4: Margin distributions of BrownBoost and RobustBoost using different ϵ on Long dataset	48
Figure 3.5: Test errors of RBA and BBA using different θ on LS with 20% noise	52
Figure 3.6: Test errors of ADB, LLB, BBA, RBA while varying number of training examples at different noise levels η	53
Figure 3.7: Margin distribution progression and potential loss function of ADB, LLB, BBA and RBA on LS with 30% label noise	55
Figure 3.8: Average ROC curves of ADB, LLB, BBA and RBA while varying training size and noise level	56
Figure 3.9: Margin distribution progression and potential loss function of ADB, LLB, BBA and RBA on Face and Satimage	57
Figure 4.1: Handwriting variations	59
Figure 4.2: Screenshots from the uRight game	64
Figure 4.3: Connectivity of various components of the uRight system	65
Figure 4.4: Web interface of uRight	66

Figure 4.5:	Example of bit-per-second (BPS) and accuracy plot over time of a user	67
Figure 4.6:	Prototypes over time for a user	68
Figure 4.7:	Mistakes made by a user	69
Figure 4.8:	Confusion Matrix for a user	70
Figure 4.9:	Average error rate as a function of the training set size and the value of the mixing parameter ρ	74
Figure 4.10:	Hidden state reduction process	76
Figure 5.1:	Handwriting recognition channel.	80
Figure 5.2:	Channel rate per session of all users	87
Figure 5.3:	Average writing time per session and the average mutual information per session under the condition R_{fixed}	88
Figure 5.4:	Average channel rates	89
Figure 5.5:	Confusion between two prototypes	90
Figure 5.6:	Conditional entropy of a “z”	91
Figure 5.7:	Three confusing handwriting examples from a single user	92
Figure 5.8:	Posterior distributions as a function of time	93
Figure 6.1:	Problem with the majority rule when N is small	95
Figure 6.2:	Results from the synthetic data experiment.	107
Figure 6.3:	Handwriting trajectories from the uRight handwriting dataset.	108
Figure 6.4:	Average error rates of MinKL and Majority for each user.	109
Figure 6.5:	Examples classified incorrectly by the majority rule but correctly by the MinKL rule	110
Figure 6.6:	Visualization of the empirical center distributions	111
Figure 6.7:	MNIST and SVHN results	112

LIST OF TABLES

Table 3.1:	Various training parameters of BrownBoost (BB) and RobustBoost (RB) using ε slightly below and above the noise level η	48
Table 3.2:	Average test error rates of ADB, LLB, BBA and RBA	51
Table 3.3:	Average area under ROC of ADB, LLB, BBA and RBA on Face and Satimage.	54
Table 6.1:	Summary of the datasets used in our experiments.	102

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Yoav Freund, for all the guidance. Also, thanks to my committee, Kamalika Chaudhuri, John Hildebrand, Bhaskar Rao and Lawrence Saul. I greatly appreciate all their time and helpful suggestions during their service on my dissertation committee.

I also would like to thank all my friends here in San Diego for all the love and care. Finally, thanks all of the coffee vendors on campus for my daily dose of caffeine to keep me awake throughout the years.

Chapter 2 is based on joint work with Evan Ettinger, Matt Jacobsen, Patrick Lai and Yoav Freund titled “Detecting, Tracking and Interacting with People in a Public Space” appearing in the 11th International Conference on Multimodal Interfaces and 6th Workshop on Machine Learning for Multimodal Interaction in 2009. The dissertation author along with Evan Ettinger were the primary investigators and authors of this paper.

Chapter 3 is based on unpublished work that is currently in submission as of the writing of this thesis. It is joint work with Evan Ettinger and Yoav Freund. The dissertation author is the primary investigator and author of this work.

Chapter 5 is based on unpublished work that is currently in submission as of the writing of this thesis. It is joint work with Yoav Freund. The dissertation author is the primary investigator and author of this work.

Chapter 6 is based on unpublished work that is currently in submission as of the writing of this thesis. It is joint work with Yoav Freund. The dissertation author is the primary investigator and author of this work.

VITA

2007	B. S. in Computer Science <i>cum laude</i> , Carnegie Mellon University, Pittsburgh
2010	M. S. in Computer Science, University of California, San Diego
2014	Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

S. Cheamanunkul, Y. Freund, “Co-adaptation in a Handwriting Recognition System”, *In submission*, 2014.

S. Cheamanunkul, Y. Freund, “Improved kNN Rule for Small Training Sets”, *In submission*, 2014.

S. Cheamanunkul, E. Ettinger, Y. Freund, “Non-convex Boosting Overcomes Random Label Noise”, *In submission*, 2014.

J. Cho, B. Benson, S. Cheamanunkul, R. Kastner, “Increased Performance of FPGA-Based Color Classification System”, *FCCM*, 2010.

S. Cheamanunkul, E. Ettinger, M. Jacobsen, P. Lai and Y. Freund, “Detecting, Tracking and Interacting with People in a Public Space”, *ICMI-MLMI*, 2009.

ABSTRACT OF THE DISSERTATION

Inference of User Intent in Adaptive Input Interfaces

by

Sunsern Cheamanunkul

Doctor of Philosophy in Computer Science

University of California, San Diego, 2014

Professor Yoav Freund, Chair

As computers are increasingly ubiquitous, it is becoming more important to make our interaction with them as easy and as efficient as possible. As alternatives to the standard duo: keyboards and mice, we have touch screens, handwriting recognition, voice recognition, gesture recognition, etc. The challenge of recognition systems is that they need to accurately interpret ambiguous signals, such as pen trajectories or speech, into discrete events, such as a sequence of letters. We call this task *inference of user intent*. Over time, as a particular user uses the interface, we would like performance of the inference to improve. There are two sides to this improvement. The first is that the user improves over time, as she learns how to speak or gesture in a less ambiguous way. The second is the adaptation of the computer, achieved through machine learning methods. We refer to this type of interaction as *co-adaptation*.

This dissertation is a study of intent inference and of co-adaptation. The study is done in the context of two systems. The first system is a touch-free interface that is set up in a public space, called The Automatic Cameraman (TAC). On TAC, we focus on the problem of real-time user engagement. The second system is an adaptive handwriting recognition system for mobile phones, called uRight. The main characteristic of this system is that it not only adapts to each individual user but also provides insightful feedback to reinforce human learning. On this system, we explore the impact of co-adaptation on the information transfer rate in the context of handwriting recognition.

Using machine learning to process and combine signals from two sensing modalities: visual and audio, TAC was able to robustly detect and track a user with minimal calibration. The engagement of the user was effectively identified by using a simple hand interaction protocol. Our experiment in co-adaptation confirmed our intuition that machine learning works best when matched with human adaptation. The concept of co-adaptation provides an interesting direction for future research on recognition systems.

Chapter 1

Introduction

Today we live in environments with ubiquitous computing. Our homes, gardens, cars, phones are all controlled by computer systems. We use a variety of modalities to interact with these computers through their *input interfaces*. For example, to change channels on an entertainment system, we press a button on a remote control. To search for a dinner recipe on the internet, we type in keywords into a web browser with a keyboard. To call a friend on a smartphone, we use a virtual keypad on a touch screen or voice commands through a voice recognition system.

Input interfaces are available in many forms and operate on various modalities. Some of them are more suitable for some applications than others. For desktop computers, mice and keyboards have been considered the golden standard as they are easy to operate and capable of achieving a high throughput for experienced users. On small form factor devices such as smartphones or tablets where space is limited, touch- and pen-based interfaces are widely used instead of mice and keyboards. In our living rooms, entertainment systems are often controlled by remote controls with only a few buttons. Voice recognition and camera-based interfaces are often used in situations where hands are already occupied in certain applications such as gaming, in-car entertainment systems, or surgical operations.

There are a number of challenges involved in designing an effective input interface. In particular, we are interested in two problems related to the inference of user intent. First, we consider the problem of identifying when a user wants to engage with the interface. We refer to this problem as the *engagement identification problem*. When

the interface is based on discrete events such as pressing a key or touching a screen, such events are commonly used as indications of the intent to engage. While this is true most of the time, it is also possible that the signals were triggered unintentionally such as when the user unknowingly leans on a keyboard or accidentally dials a smartphone while walking. On the other hand, for interfaces that lack such discrete events such as voice- or visual-based interfaces, the problem can be quite difficult especially when the environment is unconstrained. A voice recognition interface may be able to detect the beginning of a speech from a single person in a quiet room but fail in a room full of people talking to each other. In human-to-human communication, we often use subtle cues such as body language and eye-contact to establish the intent to engage with one another, but it is difficult for computers to perceive and interpret such signals.

After the user is engaged with the interface, the next problem to consider is how to interpret the information transmitted by the user. Specifically, the input interface must be able to recover the message, or information, from the user so that the rest of the system can respond appropriately. In case of keyboards, information is encoded into a sequence of discrete key-presses from which the original message can be trivially inferred. However, when the information is encoded into complex continuous signals such as pen trajectories (in pen-based interfaces) or speech (in voice recognition), the inference problem becomes much more difficult due to noise and variety of the input signals.

An important quantity to consider in this context is the amount of information successfully transferred over a time period. It is also known as the *information transfer rate*, which is measured in bits per second, regardless of the communication modality. Consider again the problem of entering texts to an entertainment system. A poor but common choice is to have an on-screen keyboard controlled by using arrow keys and a few buttons on a typical remote control. This method might be intuitive for novice users but it performs poorly in terms of the information transfer rate. An alternative approach is to include a keyboard on the remote control. Although this approach allows the user to enter text more quickly than the previous approach, it results in a remote control with a large number of buttons that can be confusing to use. The contention between the information transfer rate and the user's effort is always present and we need to find a

balance between them in order to design an efficient input interface.

This dissertation surrounds our machine learning-based solutions to the two inference problems in the context of two systems: a multimodal, touch-free interface called the Automatic Cameraman (TAC) and an adaptive handwriting recognition system called uRight. We will introduce each system in its own section below.

1.1 The Automatic Cameraman (TAC)

The first part of this dissertation is centered around the development of a multimodal, touch-free interface that allows people to interact with a computer system through speech and gestures. We followed the “smart room” approach where a camera and a set of microphones are used to capture users’ speech and gestures, and built an autonomous system that engages people passing through a public space who are not immediately aware of the system’s presence. We call this system, *the Automatic Cameraman (TAC)* as its main task is to interact with the users allowing them to record videos of themselves through speech and gestures.

One of the challenges we have to contend with when placing a system in a public space is the many different interaction scenarios that can occur within the environment. First, the number and location of users are unconstrained. Second, the lighting and sound conditions are much worse than those that can be achieved in a controlled laboratory, and even worse, these conditions can vary with time. Therefore, a large amount of calibration and re-calibration is required to get to a level of acceptable accuracy and reliability. In our system, we use machine learning methods to make our system adaptive to its environment and user-base.

In order to use the machine learning methods, we need to collect large amounts of training data. This brings us to one of the inherent problems in developing systems that are based on machine learning: on the one hand, one needs training data in order to train the system, but on the other hand, one needs an operational system in order to collect data. Our solution to this cyclical problem is to adopt an evolutionary approach to system development. We first used a minimalistic calibration process to get the system roughly working. At first, the accuracy of the system was poor and it took significant

effort to automatically record any useful data. Once we managed to record a sufficient amount of data, we retrained the system, which improved its accuracy and allowed us to collect useful data more easily. With larger corpus of more useful video recordings, we were also able to add new features such as face recognition, making the interaction more interesting and engaging.

To identify whether the users want to engage the system or just passing by, we developed into TAC a protocol for the user to start and stop video recording by putting their hands in and out of an on-screen button. The protocol is initiated by a user calling out the system. Based on audio input to the microphone array, TAC determines the origin of the sound and points the pan-tilt-zoom (PTZ) camera roughly in the direction. Next, based on visual input, it locates and centers the camera at the user's face. Then, it shows an on-screen button that requires skin-color to activate. The user needs to place one of their hand inside the on-screen button, take it out, and then back inside again to start the recording. To stop the recording, the user will need to place their hand inside the on-screen again. The benefits of this protocol are twofold. First, it serves as a consent for TAC to record them. Second, it is to ensure that the user really wants to engage the system. We will discuss the technical details in Chapter 2.

1.2 The uRight System

One of the challenges brought on by the miniaturization of mobile computing devices such as smart phones and tablets is the difficulty of entering information into the device. The communication bandwidth from the device to the human, utilizing high-resolution screens and high fidelity sound, is very high. However, the bandwidth from the human to the device is severely constrained by the size of our fingers and by the difficulty of performing voice recognition in noisy environments.

As the screen real estate becomes smaller, the standard soft-keyboard can only fit up to about 40 fingertip-sized keys on one screen. While 40 keys are sufficient for languages with a small set of characters such as English, it is not suitable for languages with more characters such as Thai or Chinese which contains more than 60 and thousands of different characters respectively. For such languages, as well as in the multilingual set-

tings, the users are required to repeatedly switch between different keyboard layouts in order to find the desired character. Many alternatives to the standard soft-keyboard exist. For example, Swype[®] provides a method for tracing a path between keyboard keys and lifting the finger from the screen at the end of each word. Dasher [GWM⁺03] is a particularly innovative method where typing is replaced by using a joystick-like pointer to fly through clouds of characters. Finally, there are handwriting recognition software that allow the user to enter information using natural handwriting.

A user of any one of these methods typically improves significantly with practice. There are many competitions between different data entry methods. However, these comparisons are inherently flawed in that the contender is always a person who can enter information faster than the current record holder, most likely as a result of extensive training. In other words, the effect of user adaptation cannot be ignored. Figure 1.1 shows the text entry rate and the approximate training time of various input methods.

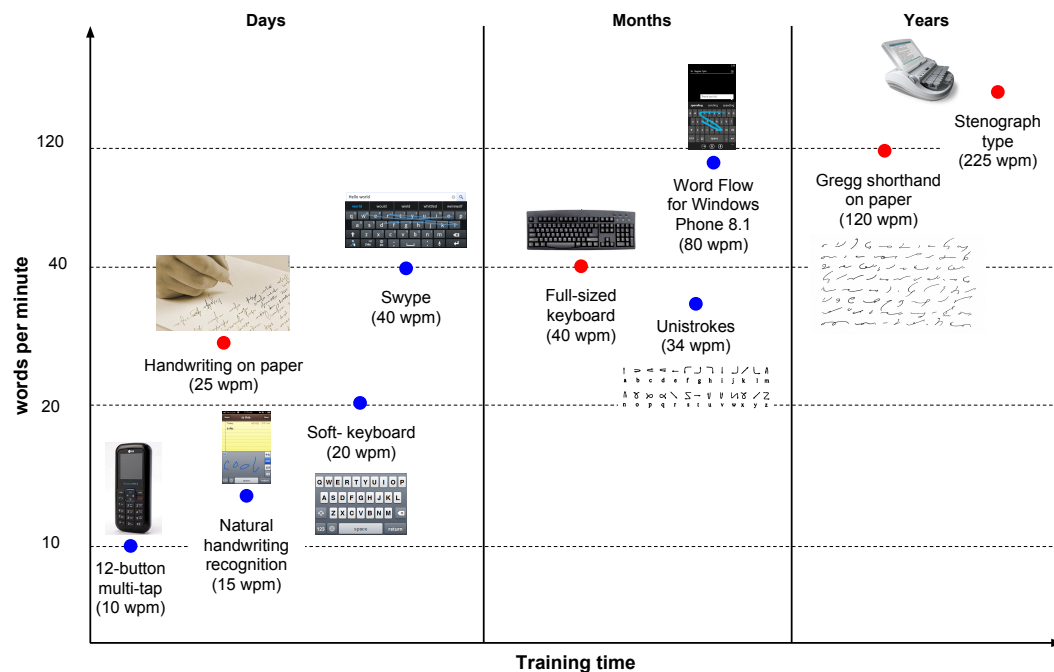


Figure 1.1: Text entry rate and training time of various methods.

As we are interested in machine learning methods, we arrive at the interesting situation in which both the computer and the human adapt over time in an effort to maximize the information transfer rate of the data entry method. We refer to this situation

as *co-adaptation*. Based on this idea, we developed a handwriting recognition system called uRight and studied the impact of co-adaptation on the information transfer rate in the context handwriting recognition, which will be discussed in detail in Chapter 4- 5.

1.3 Thesis Organization

This dissertation is organized as follows. In Chapter 2, we address the engagement identification problem on TAC, which is to identifying when the user wants to interact. We explicitly outline the user interaction protocol of TAC and the design and implementation of the underlying face localization unit that includes a skin-color detector, a face detector and a face tracker. In addition to the face localization unit, we describe our approach to face recognition implemented on TAC.

During the development of our face detector for TAC, we found an interesting situation where some of face images in our training set were incorrectly labeled, causing negative impact on the classification accuracy of traditional boosting algorithm such as AdaBoost and LogitBoost. In Chapter 3, we investigate two non-convex boosting algorithms: BrownBoost and RobustBoost and present empirical evidence showing that the two boosting algorithms are less sensitive to the label noise than AdaBoost and LogitBoost.

In Chapter 4 we describe our design philosophy and the architecture of a co-adaptive handwriting recognition system called uRight. We also outline the algorithms for training and adapting the handwriting recognizer.

In Chapter 5, we devise an information-theoretic framework for quantifying the efficiency of a handwriting recognition system that considers both the user and the computer as a single system. We use this framework to characterize the impact of machine adaptation and of human adaptation based on empirical data from a small deployment.

In Chapter 6, we present a simple k -NN rule that incorporates non-majority classes into the prediction and show that, by using this new rule, we achieve lower error rates compared to the traditional majority rule in many datasets including our handwriting recognition dataset.

Finally, in Chapter 7 we give overall conclusions of our studies and suggest directions for future work.

Chapter 2

User Engagement Identification on the Automatic Cameraman (TAC)

In this chapter, we discuss the problem of user engagement identification on Automatic Cameraman (TAC) with an emphasis on the design and implementation of the face localization unit, which enables TAC to detect, locate and track the user by using visual input from a pan-tilt-zoom (PTZ) video camera.

The chapter is organized as follows. First, in Section 2.1, we briefly describe the overall system architecture of TAC. In Section 2.2, we describe how TAC interacts with a user. Finally, we describe our implementation of the face localization on TAC in Section 2.3.

2.1 System Architecture

The Automatic Cameraman (TAC) is a touch-free interactive system setup for recording self-videos, located on the 4th floor of the Computer Science and Engineering building (EBU-3b) at UC San Diego. The display consists of four 52" plasma TVs set up in a 2x2 grid. TAC uses two sensing modalities: a pan-tilt-zoom (PTZ) video camera and an array of 7 microphones. The PTZ camera is mounted at the top-middle of the display. Figure 2.1 shows the display configuration and the PTZ camera. The microphones are mounted at different locations: one at each corner of the display and another three on the ceiling. The system is set up in the public lobby where there are



Figure 2.1: A frontal view of TAC display unit. The position of the PTZ camera and four of the microphones are shown.

traffic coming from either the elevators or the stairwells. Figure 2.2 depicts a sketch of the lobby in which TAC is set up. In the lobby, there is a large floor-to-ceiling window to the left of the display that changes the lighting of the room depending on the position of the sun and the weather condition.

The signals from the PTZ camera and the microphone array are digitized and fed into 2.66 Ghz quad-core G4 Mac located on the other side of the wall from the display. The audio signal is processed by the audio localization unit that is responsible for locating the source. The video signal is processed by the face localization unit. To interact with the user, TAC utilizes the outputs from these two units.

TAC is implemented in Max 5 graphical programming environment¹. Max allows for quick and simple development of signal processing applications by a simple graphical programming model. At a high level, programming in Max involves connecting functional units, called *patches* together with edges that represents signal traffic. Each functional unit can be implemented separately using more conventional languages such as C and Java. The audio localization patch is written in C while the face localiza-

¹ Available from <http://cycling74.com/products/max/>

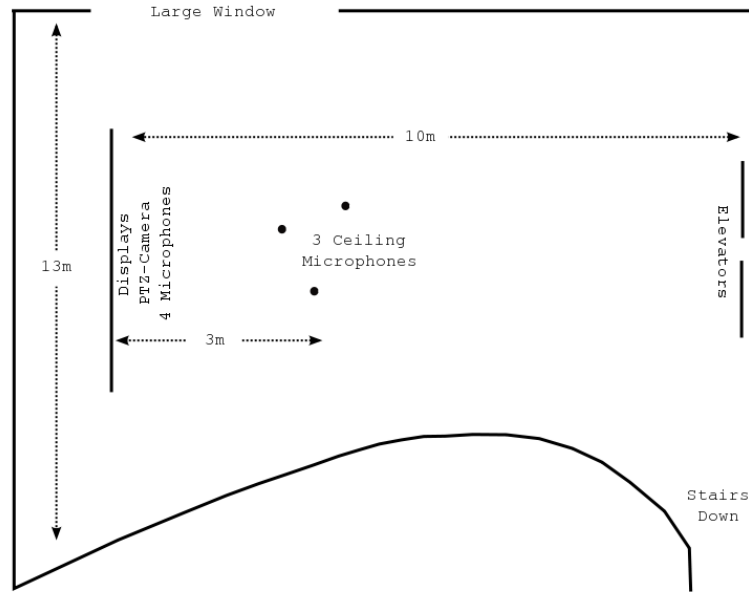


Figure 2.2: A layout of the lobby area in which TAC is set up.

tion patch is written in Java.

2.2 User Engagement Identification

Since TAC is set up in a public area where people either want to engage with the system or ignore it completely, it is important for TAC to identify those who want to engage effectively. Our approach is to use machine learning methods to process and combine inputs from two sensing modalities: audio and video, allowing the system to detect and track the user in real-time. To determine the user engagement, we implement a simple hand interaction protocol based on skin-color detection. Note that it is not enough to just detect a user. Ultimately, we want to determine the intent of the user – whether not the user wants to engage with the system. The hand interaction protocol serves as a confirmation of the intent.

An interaction with TAC is initiated by a user calling out to TAC. Using the audio localization, TAC determines the origin of the sound based on *time delay of arrival* (TDOA) [Ett10] and issues an appropriate pan-tilt commands to point the camera

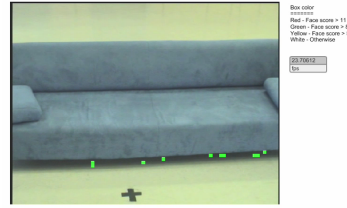
roughly at the location. Next, it scans for a front-facing face in the area using the face localization unit, which will be discussed in detail in Section 2.3. This step is important as audio signal alone often leads to false detection due to various noise in the environment such as the elevator sound, the sound from people taking on the phone or even footsteps. When it finds a face, additional pan-tilt-zoom commands are issued to the camera so that the face is positioned at the center of the frame and has size roughly about 200-by-200 pixels. After the face is detected, it is continuously tracked by the face tracking unit on TAC. At this stage, the user can control the recording, namely start and stop recording, by placing one of their hands on an on-screen button. The activation of the button is based on the amount of skin-color in the button area. Figure 2.3 illustrates an interaction with TAC.

To start a recording, the user is instructed to place one of their hand inside an on-screen button, move outside and back inside again. This hand-in-out-in sequence is designed to ensure that the user wants to engage the system and to minimize false detection. To stop the recording, we again require the user to put their hand on another on-screen button placed at a different location than the start button to prevent an immediate stopping of the recording when the user holds their hand on the start button for too long.

2.3 Face Localization

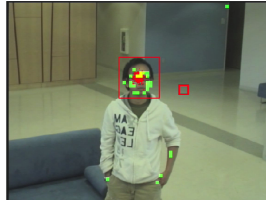
The face localization unit consists of three major components: a skin-color detector, a face detector and a face tracker. The skin-color detector operates at pixel level and determines whether a pixel has the color of human skin. The face detector is responsible for producing candidate face locations along with their likelihood scores in a specific region of the frame. Finally, the face tracker combines the current candidate face locations from the face detector with the temporal information from previous frames to determine a single face location for the current frame. It also controls the region in which the face detector scans for candidates. Figure 2.4 shows the connections between the components in the face localization unit.

We take machine learning approach in designing both the skin-color detector and



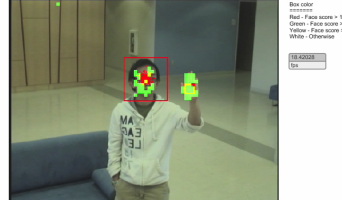
Call me and I'll look at you...

(a) TAC resting



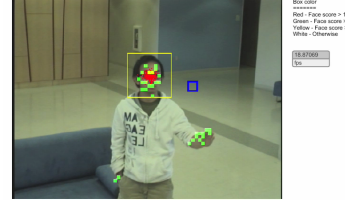
To record yourself, put your hand in the red box.

(b) Locate speaker



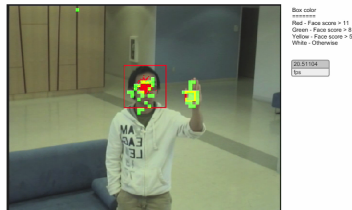
To record yourself, put your hand in the red box.

(c) Put hand on button



Now take your hand out of the box.

(d) Take hand off



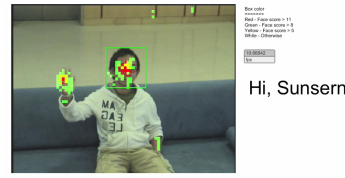
okay, now put your hand in the box again...

(e) Onto button again



Please sit down on the couch.
To stop recording, place your hand on the stop button.

(f) Recording starts



Please sit down on the couch.
To stop recording, place your hand on the stop button.

(g) Stop recording

Figure 2.3: User interaction with TAC.

the face detector. Specifically, the detectors are trained using boosting [SF12] on a training set collected from the actual setup of TAC. The key benefit of this approach is that it is calibration-free as the parameters are learned automatically through machine learning. However, this approach requires labeled examples which are typically expensive to collect in a large number. To reduce the amount of labeling, we adopt an active learning technique [CAL94], which excels in the situation where the unlabeled data is abundant and the labeled data is expensive. In particular, our training algorithm is based on the margin-based active learning algorithm proposed in [BBZ07], which fits naturally into the boosting framework.

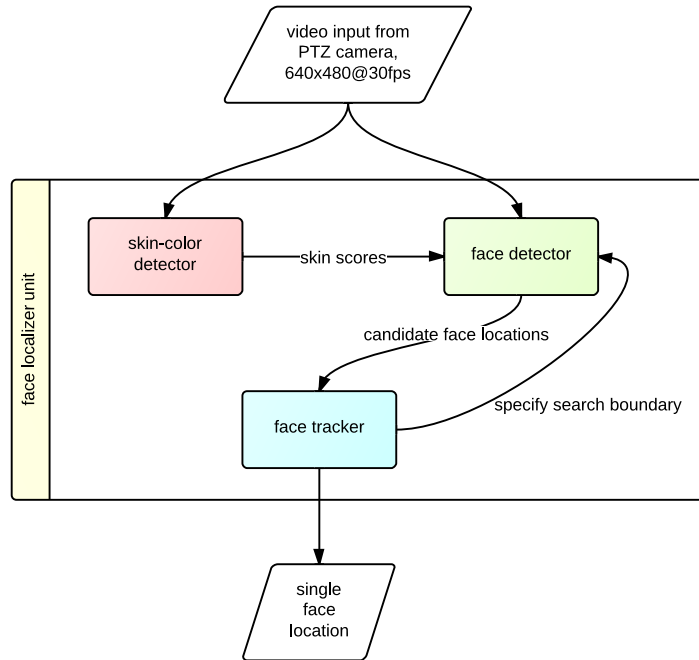


Figure 2.4: Face localization unit on TAC.

2.3.1 Margin-based Active Training Methodology

We first briefly review the notion of classification margin and score in context of boosting. Recall that boosting is a learning algorithm that combines many “weak” rules to form a “strong” or final classifier. More precisely, let (x, y) , with $y \in \{-1, +1\}$ denote a labeled example. Let $h_i : X \rightarrow \{-1, +1\}$ denote the weak rules. Then, the output of a boosting algorithm is a rule of the form

$$F(x) = \text{sign} \left(\sum_i \alpha_i h_i(x) \right)$$

Following the notation in [SFBL98], the classification score of an example is defined as:

$$s(x) = \vec{\alpha} \cdot \vec{h}(x)$$

and the classification margin of an example is defined as:

$$m(x, y) = y \vec{\alpha} \cdot \vec{h}(x)$$

where $\vec{\alpha}$ and \vec{h} defined in the natural way. It follows that $m(x,y) > 0$ if and only if the classification rule is correct on the example (x,y) . Intuitively, the notion of margin is related to the notion of confidence of the classifier. When the margin of an example is high, the classifier is more confident about its prediction of that example. The examples with small margins are considered “hard” for the classifier. These “hard” examples are crucial to improving the performance of the classifier as the learning algorithm focuses more on them rather than “easy” examples i.e examples with large margins. This idea is the basis of the margin-based active training methodology. Figure 2.5 shows a score distribution with low- and high-margin regions indicated.

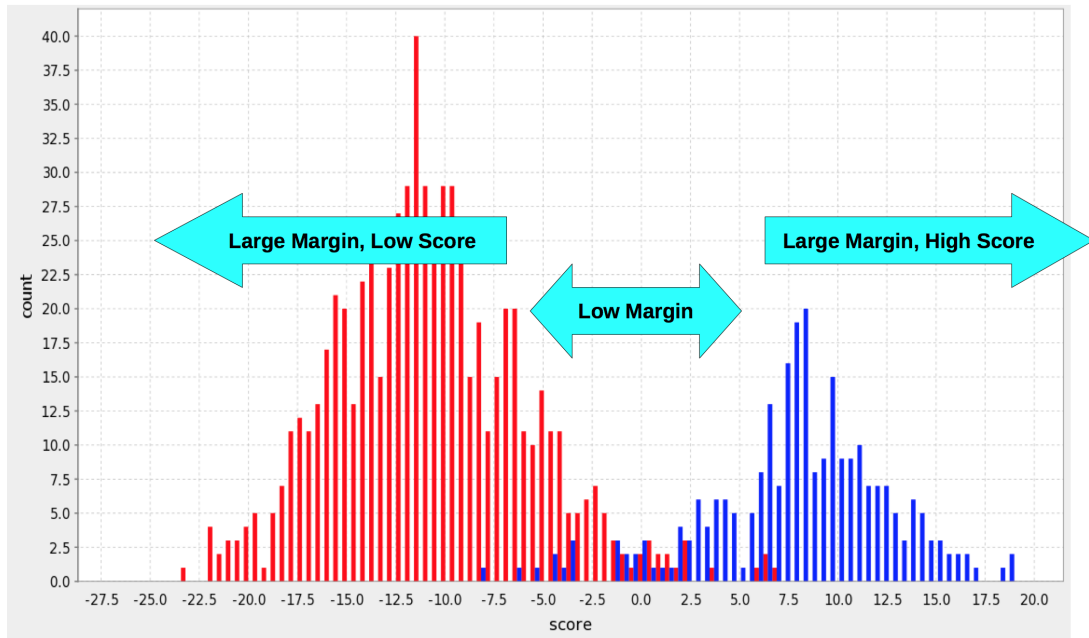


Figure 2.5: Example of boosting score distribution. The positive class is shown in blue and the negative class is shown in red.

Our training algorithm is based on the margin-based active learning algorithm proposed and analyzed by Balcan et al. in [BBZ07]. The training process happens in rounds. In the first round, we create a set of labeled examples by sampling from the pool of unlabeled examples and labeling them. Let \mathcal{D}_0 denote the initial training set. Using \mathcal{D}_0 , we train a boosted classifier C_0 . In round k , we run C_{k-1} on a new set of unlabeled examples. Using the notion of classification score, we partition the examples into 3 groups with respect to their scores. The first group consists of examples with high

Algorithm 1 Margin-based active training algorithm

Input: The initial training set \mathcal{D}_0 , the number of training rounds R .

Output: The trained classifier C_R .

- 1: **for** $k = 1$ **to** R **do**
 - 2: Train a classifier C_k using \mathcal{D}_{k-1} as the training set.
 - 3: Run C_k on a new set of unlabeled examples and partition the examples into 3 groups:
 - **High positive:** examples with $s(x) > \theta$.
 - **High negative:** examples with $s(x) < -\theta$.
 - **Unsure:** examples with $-\theta \leq s(x) \leq \theta$.
 - 4: Create \mathcal{D}_k by combining \mathcal{D}_{k-1} with samples from the high-positive group and the negative-group labeled positive and negative respectively, and with examples in the unsure group labeled manually.
 - 5: **end for**
-

positive scores $s(x) > \theta$. These examples corresponds to what C_{k-1} classifies as positive with high confidence. The second group consists of examples with low negative scores $s(x) < -\theta$. These examples corresponds to what C_{k-1} classifies as negative with high confidence. Lastly, the third group consists of examples whose score is close to zero, $-\theta \leq s(x) \leq \theta$. The examples in this group are considered “unsure” by C_{k-1} .

After the unlabeled examples are partitioned, a new training set \mathcal{D}_k is created by adding new examples to \mathcal{D}_{k-1} in the following fashion. First, with a small probability p , we sample the examples from the first group and the second group, label them as positive and negative respectively, and add them to the new training set. Next, we look at the examples in the “unsure” group, label them and add them to the new training set. The training algorithm is summarized in Algorithm 1.

2.3.2 Skin-color Detector

In TAC, skin color information is used as a low-level feature for face detection and as a control mechanism for on-screen buttons. We design a simple, yet effective skin-color detector that identifies each pixel as either skin or non-skin. Per our design goals, we want our skin-color detector to be efficient and adaptive to TAC’s user-base and lighting conditions.

In order to minimize the response time, we avoid using any high-level contexts as detection features. Instead, the detector operates solely at the pixel level, in particular in the HSV color space. One of the challenges of a pixel-based skin-color detector is that pixel values of the same object vary tremendously depending on the environment e.g. lighting conditions, camera model, etc. Thus, learning simple ranges on each color channel does not work well in practice. Instead, we train our detector using a discriminative approach, and by doing so, we make no explicit assumptions about the distribution of skin color in color space. In addition, since the detector is trained from actual video collected by TAC, the resulting detector is specialized to how skin appears in our given environment.

Technically, our skin-color detector is a boosted binary classifier. In addition to the binary prediction, it also gives a likelihood score referred to as “skin score” which is essentially the boosting score defined in Section 2.3.1.

As for training, we follow the training methodology described in Section 2.3.1. The training data is a set of 320x240 RGB images extracted from the early videos recorded by TAC. For labeling, we use a software called Digital Notebook ² to manually annotate regions of each training image as skin and non-skin. The pixels in the annotated regions are then sampled to create the training set. We use the hue, saturation and volume (HSV) representation of the pixel as features in the boosting process. Figure 2.6 shows examples of annotated images.

We run AdaBoost [FS96, FS99] with decision stumps as based rules. We use the implementation of AdaBoost available from the JBoost package ³. The output classifier is in the format of an alternating decision tree (ADT) [FM99]. Briefly, an ADTree is

²Available from <http://www.bioimage.ucsb.edu/Digital%20Notebook>

³Available from <http://jboost.sourceforge.net/>

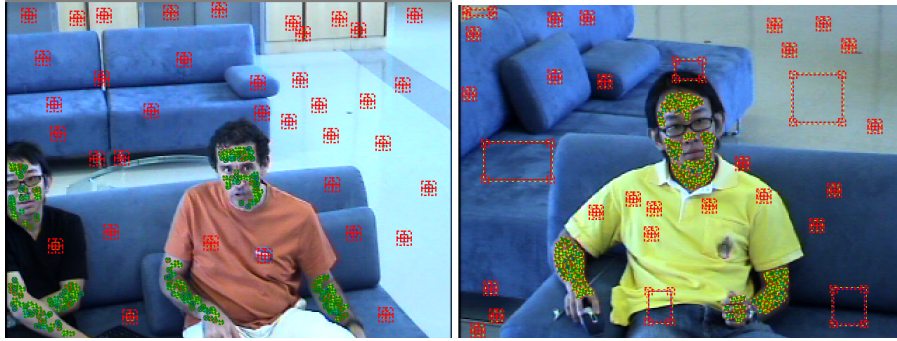


Figure 2.6: Annotated images for skin detection. Skin and non-skin pixels are indicated by green regions and red regions respectively.

a generalized decision tree that has tree levels that alternate between decision nodes and prediction nodes. Decision nodes are exactly the same as those in a decision tree, typically a threshold on an individual feature. The prediction nodes give a real valued prediction which in the boosting terminology correspond to the α values associated with each weak hypotheses. Unlike a decision tree where examples follow a single path in the tree, in an ADTree whenever an example encounters a prediction node, it traverses to all the children decision nodes below it. By changing the growing procedure during learning one can emulate traditional boosted stumps or something very similar to boosted decision trees.

The results of the skin-color detector after the first iteration of training is shown in Figure 2.7. The detection gets better as we retrain the detector in the active training fashion. Figure 2.8 show the improvement from several iterations of active training.

The detection results can be further improved by applying a smoothing filter over the neighboring pixels. Specifically, we determine the label of a pixel by performing a majority vote among neighboring pixels. The results of the skin-color detector with and without voting scheme are shown in Figure 2.9.

On TAC, our Java implementation of the trained detector operates at the rate of 25 frames per seconds at the resolution of 320x240. Although this performance is sufficient for TAC, the performance can be greatly increased when implemented on field-programable gate array (FPGA). The FPGA implementation of the same detector can achieve up to 233 frames per second at 640x480 [CBCK10].

Aside from being used as low-level features for face detection, the output of the

skin-color detector is also used for “on-screen” buttons on TAC. The on-screen buttons are buttons that appear spliced into the video displayed on one of the displays similar to “hot spot” work of [JL05]. TAC uses these buttons as a means to interact with the user. A user can simply place their hand hovering over the virtual buttons in order to activate them. In the current implementation, the system uses one of these buttons to get authorization from the user before recording a video. The activation of each buttons is determined by the amount of skin pixels contained within the button’s boundaries. When a user places their hand on the button, the number of skin pixels in the button area is counted and the button is activated whenever this number is greater than some threshold. Figure 2.10 shows the on-screen button used for starting and stopping recording on TAC.

2.3.3 Face Detector

The face detector for TAC is based on the sliding window approach in the Viola-Jones face detector [VJ01]. The main difference between the Viola-Jones face detector and ours is the feature set. Instead of Haar-like features, our classification features are based on the histogram of skin scores produced by the skin detector and the histogram of gradients (HOG) [DT05], which is popular in visual object detection tasks. Figure 2.11 shows images of the skin scores and the gradients and Figure 2.12 shows the mask used to extract a feature vector from a square image patch. In each region of the mask, a histogram of skin scores and of HOGs are calculated and concatenated into a single feature vector.

Similar to the skin-color detector, the face detector is also a boosted binary classifier. The classifier is trained using LogLossBoost, which is a LogitBoost [FHT98] implementation in JBoost. The output classifier is also an alternating decision tree (ADTree).

We use margin-based active learning approach discussed in Section 2.3.1 to reduce amount of labeling needed. In the first round of training, the OpenCV face detector was used on the recordings from TAC to extract square patches of images that contain a face, giving us an initial set of positive examples. Randomly selected patches from TAC’s video are added as negative examples. After training an initial detector, we then used it on new videos from TAC, adding new training examples to the training set ac-

ording to the following three rules: (1) if the patch has high boosting score it is assumed to be a face and added as a positive example, (2) if the patch has large negative score it assumed a non-face and is added as a negative example, and (3) if it has score near zero then an operator must hand label it as face or non-face. After completing this process on a few new videos, the face detector was re-trained using the new training set. Since many examples are automatically labeled by the previous round's detector, the labeling workload is significantly reduced compared to labeling all examples. Figure 2.13 shows score distributions over multiple iteration of training and Figure 2.14 shows examples of face images categorized by their classification scores.

When the detector is deployed, one major computational bottleneck is in calculating the feature vector for each image patch. In particular, we need to compute skin scores for each pixel, HOG values, and collect their histograms within each region of the detection mask. Normally, we need to do this for every location-size pair in the image, quickly becoming very computationally expensive. If the face was detected in a previous frame we can use tracking which is described in the next section. The computational time for a full scan can be reduced greatly using the integral image technique presented in [VJ01]. This technique leverages a nice mathematical trick to avoid the repeated computation of feature values across patches and quickly returning the histogram values needed for transforming an image patch into a feature vector for input into the boosted face detector.

2.3.4 Face Tracker

In principle, face localization can be realized solely by running full-frame face detection at every frame. However, this solution is too computational expensive. On TAC, our full-frame face detection needs to calculate scores of about 75000 candidate windows. At the maximum, it can run at the rate of only 1-2 frame per second. As the number of candidate windows at which the detections happen is fixed, the detections are of limited accuracy and jittery. A better use of resources is to use an adaptive grid where the range of locations and sizes that are measured is based on detections at previous frames. We instead use a tracking algorithm to combine detections from the face detector.

Translating the output of the detector into predictions of the location of a face requires solving two related problems. The first problem is peak-finding, in practice, a reliable detector would not detect a face only in a single location and size but in a range of locations and sizes. It is therefore necessary to identify the location of the “best” detection. Using the location with the maximal score is reasonable, but results in a very noisy and jittery detection. It is better to smooth the scores before finding the maximum, but then the question becomes how much to smooth. The second problem is that of tracking, i.e. taking into account that the face is likely to either stay at one location or else move at slowly varying speed (recall that the location of our PTZ camera is fixed and that the pan and tilt are known and can be subtracted away).

Our solution to the above problems is to apply a tracking algorithm to take into account the dynamics of the face. In particular, we use a variant of particle filters [AMGC02] called Normal Hedge tracking algorithm proposed by Chaudhuri et al. in [CFH10]. The Normal Hedge tracking algorithm is derived from a parameter-free online algorithm for hedging over the predictions from a group of N experts that has strong theoretical guarantees [CFH09]. The algorithm is summarized in Algorithm 2.

The speed-up obtained by using the tracker is significant over using face detection alone without tracking. Without the tracking algorithm, we need to calculate the score of 75,000 boxes in the video frame. As a result, we can perform face detection only 2-3 times per second and the resolution of our detections is not very high, resulting in jittery behavior. On the other hand, when we use our variant of the tracking algorithm, we calculate scores for only 500 boxes per frame. As a result we can track at 30 frames per second, which is the rate of the video camera, and as the locations of the detection boxes is adaptive, we get a much smoother and less jittery detection. On the other hand, the tracker can get stuck in local maxima, i.e. locations that are not faces but seem similar to faces and have the correct dynamics. We therefore perform a complete scan of the video frame every second to detect the actual location of the face and get the tracker out of local maxima. Figure 2.15 shows a screen capture of the tracker in action.

Algorithm 2 Normal Hedge tracking algorithm

Initial Assumptions: At time $t - 1$, we have the set of particles $x_{t-1}^{(i)}$ and Normal Hedge weights $w_{t-1}^{(i)}$, $i \in \{1, \dots, m\}$

- 1: **Regret Update:** Obtain losses $\ell_t^{(i)}$ for each particle and update *discounted cumulative regrets* $R_t^{(i)} = (1 - \alpha)R_{t-1}^{(i)} + (\ell_t^A - \ell_t^{(i)})$.
 - 2: **Resample:** For each particle $x_t^{(i)}$ with $R_t^{(i)} < 0$, resample a new particle in its place
 1. Choose a current particle $x_t^{(k)}$ according to $\{w_t^{(i)}\}_{i=1}^m$.
 2. Let the new particle $x_t^{(i)} = x_t^{(k)} + u_k$, where u_k is Gaussian noise.
 3. Assign $R_t^{(i)} = (1 - \alpha)R_t^{(k)} + (\ell_t^A - \ell(x_t^{(i)}))$.
 - 3: **Weight Updates:** Update the weights of each particle by finding $c_t > 0$ that satisfies $\frac{1}{N} \sum_{i=1}^N \exp\left(\frac{([R_t^{(i)}]_+)^2}{2c_t}\right) = e$. Then, update weight distribution for round $t + 1$ by $w_{t+1}^{(i)} = \frac{[R_t^{(i)}]_+}{c_t} \exp\left(\frac{([R_t^{(i)}]_+)^2}{2c_t}\right)$. Normalize the weights so they sum to one.
 - 4: **Dynamics:** Propagate the particles through the transition equation $x_t^{(i)} = g(x_{t-1}^{(i)})$.
-

2.4 Face Recognition

Once TAC has localized a face, the next step is to identify the person. The face recognition algorithm on TAC is based on the popular notion of eigenfaces [SK87, TP91]. This method has proven to work well when the face is captured in a frontal view and carefully registered. On TAC, we perform face recognition when these conditions hold true, which hold quite often since we are engaging users causing them to look directly at the camera.

To detect whether the face is frontal and registered, we first convolve the region around the face detection with a face template. The template face is obtained by averaging over the frontal and registered faces that the system has seen thus far. TAC will try to recognize a detected face only when this correlation is high.

After a registered face is obtained, TAC computes its projection onto each eigenface and feeds them into a series of boosted binary classifiers. Each binary classifier gives a score corresponding to how likely the detected face comes from a particular per-

son in the TAC database. A person in the database is recognized when the score from his/her boosted classifier are consistently high for several seconds. Since our user-base is not too large, this scheme works well to recognize regular users and further engage them in the interaction process.

TAC updates its database nightly. This includes adding newly recognized people to its database, updating the average face, updating the eigenfaces, and retraining the per-person classifiers.

To show how well the face recognizer improves as users interact with the system more and more, we performed the following analysis: for three frequent users, we took 25 different video recordings of each user interacting with the system. Among the 25 recordings, we randomly selected 5 recordings for testing and used the rest for training. The training videos were ordered, and initially the training set consisted of only the registered faces from the first video. Subsequently, the training set was grown incrementally by adding new registered faces from the next video. In doing so, we simulated repeated interactions from the user with TAC over time. The results of 3 different users is shown in Figure 2.17. The error rate of the face recognizer goes down as the number of training sessions increases. This indicates the improvement of the face recognizer as more data is collected over time. Moreover, the final error of each classifier is less than 10% which allows for accurate recognition of these users over multiple frames of video.

Acknowledgements

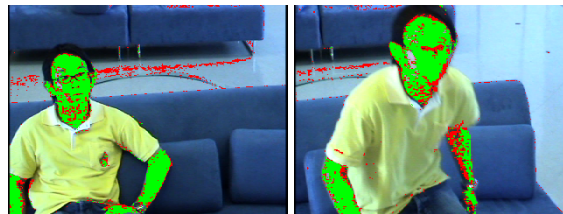
This chapter is based on joint work with Evan Ettinger, Matt Jacobsen, Patrick Lai and Yoav Freund titled “Detecting, Tracking and Interacting with People in a Public Space” appearing in the 11th International Conference on Multimodal Interfaces and 6th Workshop on Machine Learning for Multimodal Interaction in 2009. The dissertation author along with Evan Ettinger were the primary investigators and authors of this paper.



Figure 2.7: Results of the skin-color detector after the first iteration of active training. Green pixels indicate skin regions with high confidence and red pixels indicate predicted skin regions, yet with lower confidence.



Figure 2.8: Results of the skin-color detector after several iterations of training. Left image is the result after 1 iteration and right image is the results after several iterations of active training. Green pixels indicate skin regions with high confidence and red pixels indicate predicted skin regions, yet with lower confidence.



(a) No voting



(b) With voting

Figure 2.9: Results of the skin-color detector with and without majority vote among neighboring pixels. Green pixels indicate skin regions with high confidence and red pixels indicate predicted skin regions, yet with lower confidence.

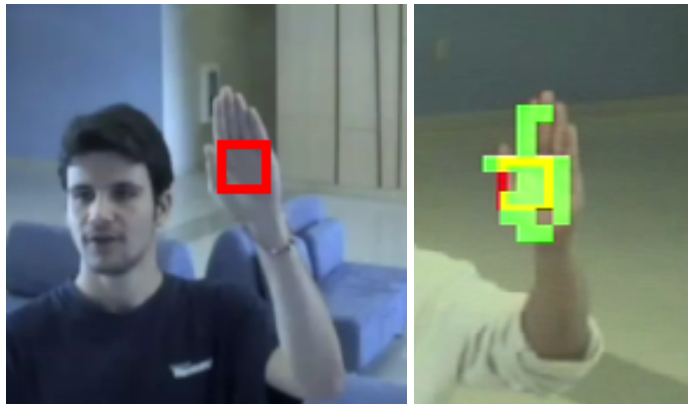
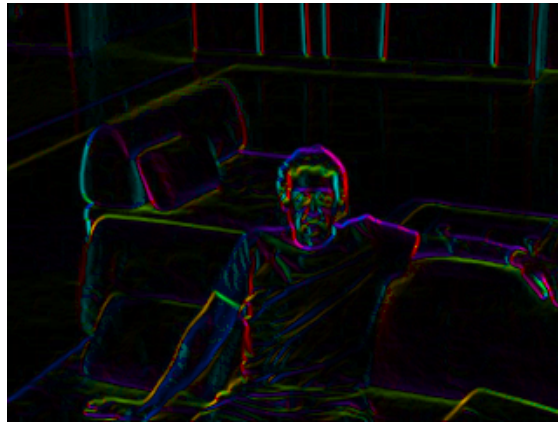


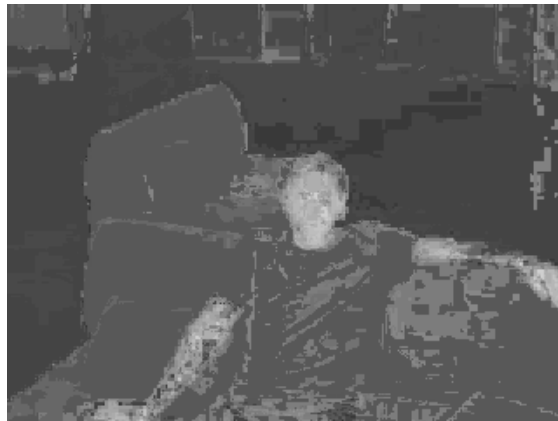
Figure 2.10: On-screen button on TAC



(a) Original image



(b) Gradients



(c) Skin

Figure 2.11: Face detection features

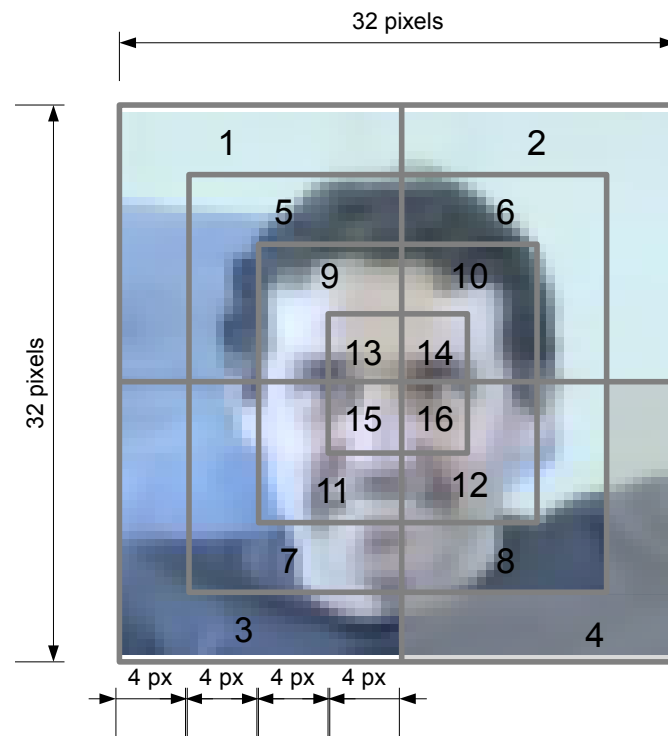
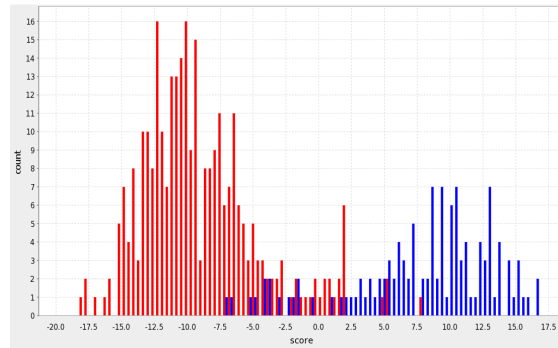
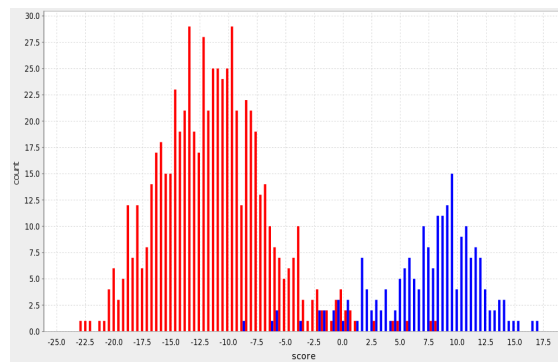


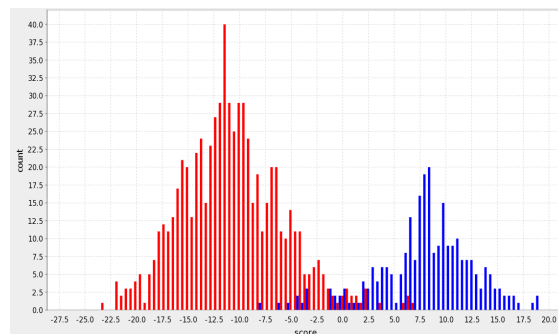
Figure 2.12: Face detection mask that defines the regions in which the skin scores and histograms of gradients are computed and combined.



(a) After iteration 1



(b) After iteration 2



(c) After iteration 3

Figure 2.13: Score distributions of the face detector over multiple iterations of training.

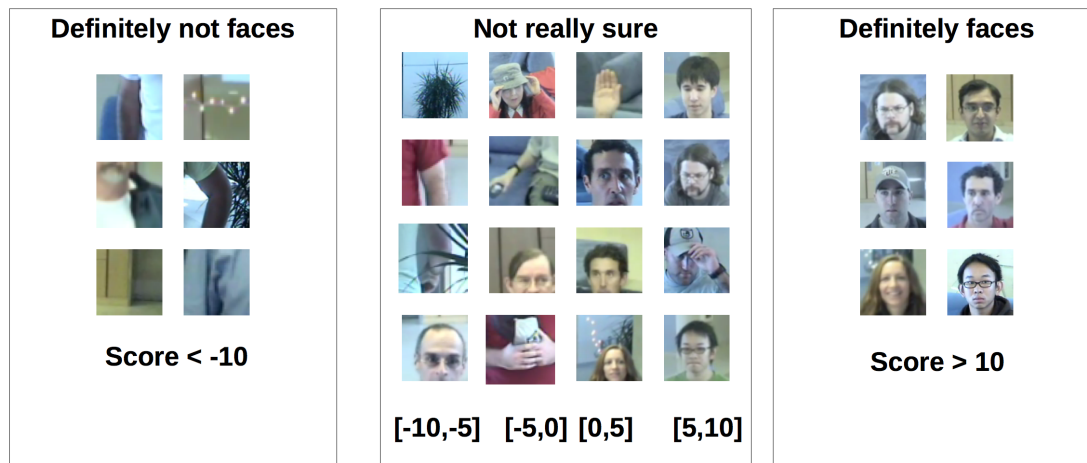


Figure 2.14: Examples of face images categorized by their classification scores.

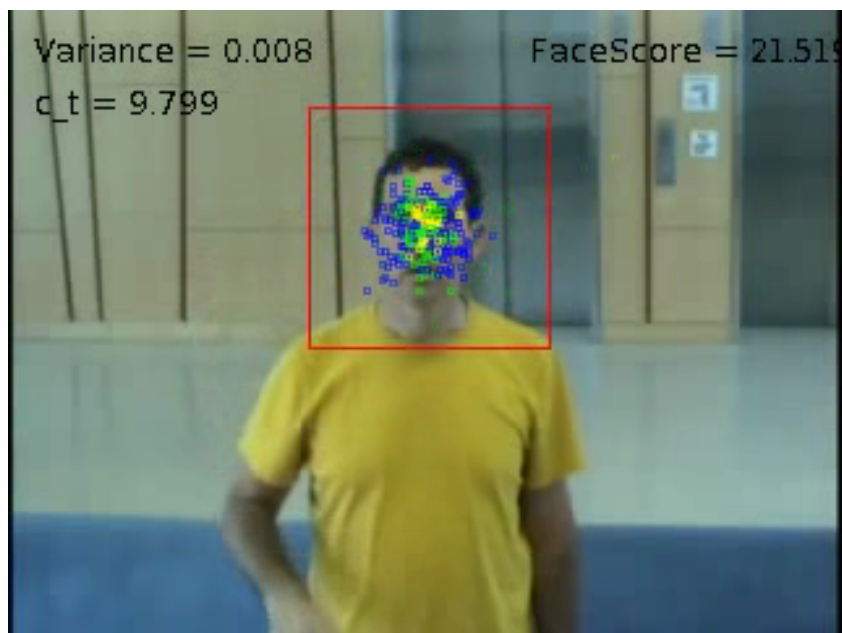


Figure 2.15: An output from the face tracker. Each dot correspond to each tracking particle and the colors indicate the current weights of the particles. Yellow particles have the highest weights, followed by green and blue respectively. The red rectangle shows the weighted average location of all particles.

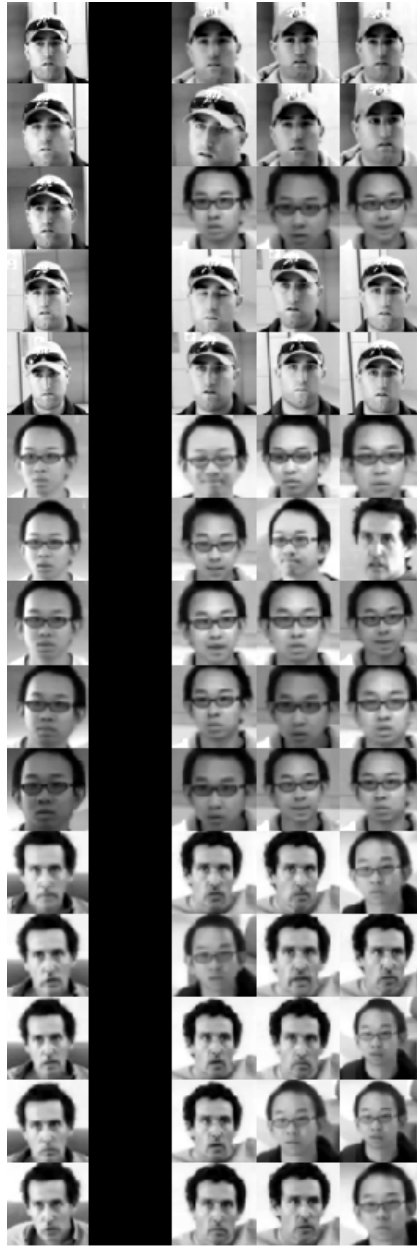


Figure 2.16: Result of the face recognition. The image in the left most column is the test image and the images in the right column are the 3 most similar faces from the training set.

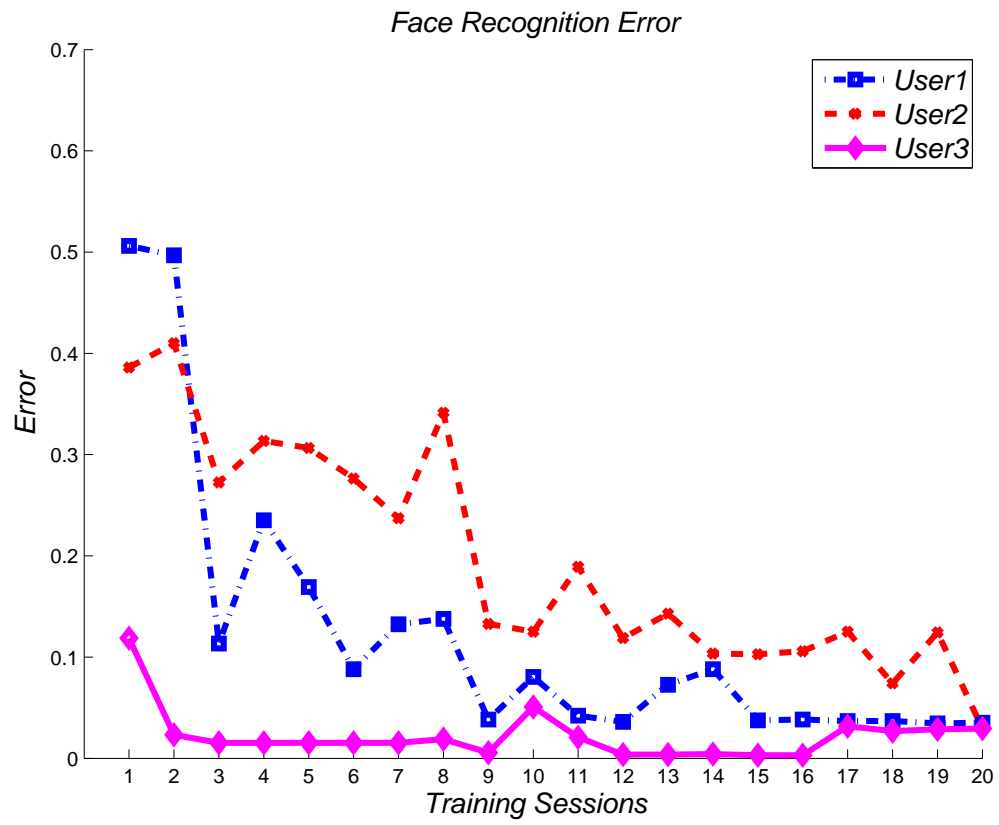


Figure 2.17: Error rate of the face recognition unit on TAC over multiple sessions.

Chapter 3

Non-convex Boosting and Random Label Noise

The sensitivity of Adaboost [FS96] to random label noise is a well-studied problem. LogitBoost [FHT98], BrownBoost and RobustBoost are boosting algorithms that claim to be less sensitive to noise than AdaBoost. In this chapter, we present the results of experiments evaluating these algorithms on both synthetic and real-world datasets. We compare the performance on each of datasets when the labels are corrupted by different levels of independent label noise. In presence of random label noise, we found that BrownBoost and RobustBoost perform significantly better than AdaBoost and LogitBoost, while the difference between each pair of algorithms is insignificant. Additionally, we provide an explanation for the difference based on the margin distributions of the algorithms.

3.1 Background and Related Work

Adaboost [SF12] is a very popular classification learning algorithm. While it is a simple and effective algorithm, the sensitivity of Adaboost to random label noise is well-documented [FS96, MO97, Die00]. The random label noise setup is one where we take a dataset for which our learning algorithm generates an accurate classifier and we flip each label in the training set with some small fixed probability. Note that the classifier that was a good classifier in the noiseless setup is still a good classifier. The

problem is that, in the noisy setup, the noisy examples mislead the learning algorithm and cause it to diverge significantly from the good classifier.

LogitBoost [FHT98] is believed to be less sensitive to random label noise than Adaboost, but it still falls prey to high levels of random labels noise. In fact, Seredio and Long [LS08] proved that, in general, any boosting algorithm that uses a convex potential function can be misled by random label noise. A number of non-convex boosting algorithms have been proposed to address the issue of random label noise. Bootkrajang and Kaban suggested a boosting algorithm whose potential loss function is based on a hyperbolic cosine [BK13]. Kalai and Kamade suggested a boosting algorithm that is based on a relabeling process [KK09]. Freund suggested a boosting algorithm, called Brownboost, that uses a *non-convex* potential function and claims to overcome random label noise. Freund suggested two boosting algorithms: Brownboost [Fre01] and RobustBoost [Fre09], that uses a *non-convex* potential function and claims to overcome random label noise. The main contribution of this chapter is experimental evidence that support this claim. The other contribution is a heuristic for automatically tuning the parameters that Brownboost and RobustBoost need as input.

Classification Margins, Convexity and Generalization

All non-recursive boosting algorithms generate a classification rule which is a thresholded linear combination of so-called “base” classification rules. More precisely, let (x, y) , with $y \in \{-1, +1\}$ denote a labeled example. Let $h_i : X \rightarrow \{-1, +1\}$ denote the base rules. then the output of the boosting algorithm is a rule of the form

$$F(x) = \text{sign} \left(\sum_i \alpha_i h_i(x) \right)$$

As it turns out, the sum which is the operand of the sign function is important for understanding the operation of boosting algorithms as well as the generalization error of the generated classifier. It is convenient to replace the sum with a dot product:

$$\sum_i \alpha_i h_i(x) = \vec{\alpha} \cdot \vec{h}(x)$$

with $\vec{\alpha}$ and \vec{h} defined in the natural way.

To characterize the relationship of the value of the sum and the label y , Schapire et al. [SFBL98] defines the “margin” of an example as:

$$m(x, y) = y\vec{\alpha} \cdot \vec{h}(x)$$

Thus $m(x, y) > 0$ if and only if the classification rule is correct on the example (x, y) . The natural goal is therefore to find base rules $\{h_i\}$ and weights $\{\alpha_i\}$ such that the number of training examples with negative margin i.e. the number of misclassified examples is minimized.

From a computational point of view, the easy case occurs when the training data is *separable*. In other words, when there exists a setting of α such that $m(x, y) > 0$ for all of the training examples. In that case, *finding* an appropriate setting for α can be done using the perceptron algorithm (see Cesa-Bianchi and Lugosi [CBL06] for a survey).

On the other hand, when the training set is not linearly separable, the problem of finding the error minimizing plane is NP-hard. We therefore have to resort to approximation techniques. The approximation used in Adaboost and Logitboost is to use a convex function that upper bounds the step function corresponding to the number of misclassifications. Specifically, Adaboost corresponds to minimizing the potential function:

$$\phi(x, y) = e^{-m(x, y)}$$

and Logitboost corresponds to minimizing the potential function

$$\phi(x, y) = \ln \left(1 + e^{-m(x, y)} \right)$$

As both of these potential functions are convex, minimizing them can be done efficiently. Figure 3.1 depicts the 0/1 error function and the potential functions corresponding to Adaboost and LogitBoost. Using a convex upper bound makes the problem tractable, but obviously there can be a significant gap between the bound and the step function which can lead to a sub-optimal solution.

Moreover, as was shown in [LS08], algorithms that minimize convex potential functions can *always* be fooled by the addition of random label noise. This naturally leads us to considering non-convex potential functions that upper bound the 0/1 error function. However, before we get to that, we first consider the question: “is minimizing the training error the right goal for a learning algorithm?”

Our ultimate goal when learning classifiers is to reduce the test error – the number of mistakes the classifier makes on the test set. As we only have access to the training data, we cannot minimize the generalization error directly. The natural goal of the algorithm is to instead minimize the training error. However, Schapire et al. [SFBL98], showed that there is a better performance measure than the performance of the boosted classifier on the training set. That is to maximize the number of training examples whose normalized margin is larger than some $\theta > 0$. Where the positive margin of the example (x, y) is defined to be

$$\hat{m}(x, y) = y \frac{\vec{\alpha} \cdot \vec{h}(x)}{\|\alpha\|_1}$$

The intuition, presented and justified in [SFBL98], is that large positive margins correspond to confident predictions. Specifically, Theorem 2 in [SFBL98] states that, with probability $1 - \delta$ over the random choice of the training set, the following inequality holds for all $\theta > 0$

$$P_D(\hat{m}(x, y) \leq 0) \leq P_S(\hat{m}(x, y) \leq \theta) + O\left(\frac{1}{\sqrt{n}} \left(\frac{d \log^2(n/d)}{\theta^2} + \log(1/\delta)\right)^{1/2}\right) \quad (3.1)$$

where P_D is the probability with respect to the true distribution, P_S is the probability with respect to the training set S whose size is n and d is the VC dimension of the base classifiers.

Note that the bound consists of two terms, the first corresponds to the fraction of the training set whose margin is at most θ and the second which is $O(1/\theta)$. The first term increases with θ while the second term decreases with θ . As the bound holds uniformly for all values of θ , we are free to choose the values of θ that would minimize the bound. Intuitively, the first term corresponds to the examples on which we “give up”. Note that giving up on an example increases the RHS of Equation 3.1 by $1/n$ regardless of amount by which the margin of the example is smaller than θ .

The goal of learning now becomes to minimize a step function that is thresholded at θ (see Figure 3.1). This does not make the problem any easier than minimizing the training error. The suggestion is, however, that minimizing this potential function will yield classifiers with smaller test error.

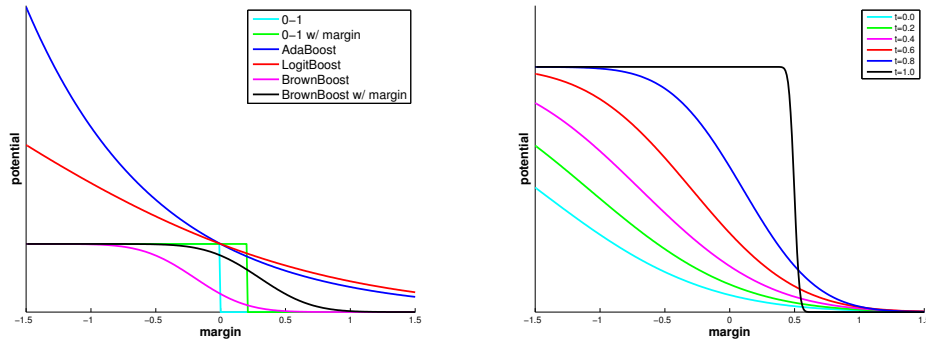


Figure 3.1: Left: Zero/one loss function with and without margin plotted with potentials for AdaBoost, LogitBoost, confidence rated BrownBoost and confidence rated RobustBoost. Right: How the potentials for BrownBoost vary with the time parameter t .

Random label noise

Consider the effect of random label noise on boosting. The weight assigned to example (x, y) by Adaboost is exponential in the margin $w(x, y) = e^{-m(x, y)}$. Suppose $c(x)$ is the best rule for the noiseless data. Suppose we now add independent label noise to the dataset. The margin of $c(x)$ on examples whose label has been flipped will be negative, resulting in a large weight being assigned to the noisy examples, resulting in base classifiers that fit the noisy examples.

3.2 BrownBoost and RobustBoost

Before we describe two non-convex boosting algorithms: BrownBoost and RobustBoost, we first discuss an algorithm called Boost-by-Majority because it provides insights into the derivation of the other two boosting algorithms.

It is worth noting that the algorithms discussed in this section are due to Freund and can be found in [Fre95, Fre01, Fre09]. This is not original work by the author of this thesis, but the fundamental ideas are central to the work that follows in the remainder of this chapter.

Boost-by-Majority

In this setup, it is useful to think of boosting as a game between two players. One player represents the boosting algorithm and is called the *weightor*. The second player represents the weak learning algorithm and is called the *chooser*. Consider a fixed training set of size n . The game proceeds in iterations that correspond to the iterations of the boosting algorithm. In each iteration, the weightor assigns weights to each of the n training examples and the learner responds by generating a weak classification rule that is slightly better than random guessing with respect to the weighted training set (has accuracy $1/2 + \gamma$ w.r.t. the weights). After T iterations the game stops and the weightor outputs the (unweighted) majority vote over all the weak rules that were generated. The weightor wins if this final rule is correct on all n training examples.

To get some intuition into this game, let us consider some simple cases. First, consider a lazy weightor that always uses the uniform distribution across examples. The adversarial response of the chooser to this strategy is simple: it chooses some weak rule which is correct on $1/2 + \gamma$ of the training examples and always outputs this same rule on each round. The majority rule is the same as this single fixed weak rule, and the weightor loses. Clearly, the weightor must alter the distribution of weights in order to prevent the chooser from always outputting the same classification rule.

As a second example, consider a lazy chooser that uses the following simple strategy: at each iteration it picks a weak rule which is correct with probability $1/2 + \gamma$ on each training example independently at random. Note that it is possible that the chooser creates a rule that is correct with less than $1/2 + \gamma$ total weight over the examples, but this can only happen with constant probability, and all the chooser needs to do is repeat this procedure until it meets this requirement. If the chooser uses this lazy strategy, then the weightor is guaranteed to win if T is sufficiently large. In fact, it is not hard to show that if the number of iterations is $O(\log(n/\epsilon)\gamma^2)$, then the probability that the majority rule is incorrect on any of the n examples is at most ϵ . Moreover, this lazy strategy is actually min-max optimal for the chooser [Fre95]. This means on the one hand there exists no strategy for the weightor to win in a smaller number of iterations, and on the other hand, there does exist a strategy for the weightor to win in exactly this number of iterations *no matter what* strategy is used by the chooser.

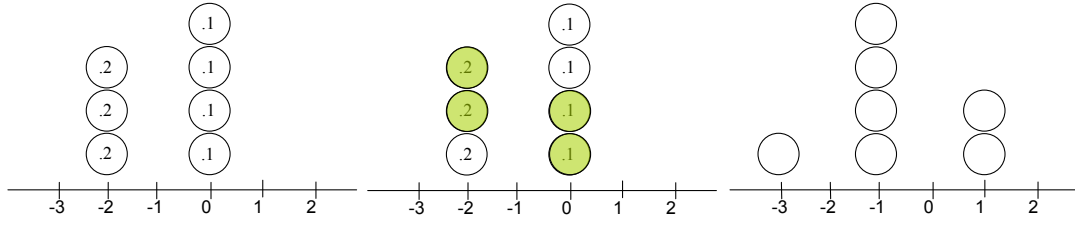


Figure 3.2: A single iteration of the chip game. Weights given by the weightor are shown inside each chip and the colored ones have been chosen by the chooser.

Let us now derive this min-max strategy for the weightor, which as a result will give us the Boost-by-Majority (BBM) algorithm. In order to simplify the description of the game, we replace the set of n training examples with a set of n chips. At any iteration t each of these chips belongs to a *bin* indexed by the integers. At the start of the game, all the chips start in bin 0. At each iteration, a weak rule is generated and those examples that were classified correctly move up a bin, while those that were incorrectly classified move down a bin. The chips that are in bin i at iteration t correspond to those examples for which the difference between the number of correct classifications (by the rules generated thus far in the game) minus the number of incorrect classifications is equal to i . We denote by n_i^t the number of chips that are in bin i at iteration t and $n_0^1 = n$ (i.e. all the chips start in bin 0).

At each round $t = 1, \dots, T$ the following steps occur:

1. The weightor assigns a weighting function to the reachable bins $w_i^t \geq 0$ for all i .
2. The chooser selects, for each bin, the number of chips that it will move upwards m_i^t . This selection must adhere to $n_i^t \geq m_i^t \geq 0$ and

$$\frac{\sum_i m_i^t w_i^t}{\sum_i n_i^t w_i^t} \geq \frac{1}{2} + \gamma$$

3. The location of the chips are updated: $n_i^{t+1} = m_{i-1}^t + (n_{i+1}^t - m_{i+1}^t)$.

See Figure 3.2 for an example of a single round of the game. The weightor wins if, at the end of the game, there are no chips in bins below bin 1. In other words, if $\sum_{i=-T}^{-1} n_i^{T+1} =$

0. We assume that T is an odd number.¹

To derive the optimal weighting strategy for the weightor, it is useful to consider the weightor's choice at the last round, round T . At this point the fate of most chips is already determined. If a chip is in a negative bin, then it is hopeless to get it to cross over to a positive bin in this last round (and vice-versa). The only chips whose fate is still to be determined are those in bin 0 (note that chips are only in even numbered bins at this last iteration since T is odd). It thus stands to reason that the weightor should put all of the weight on that bin i.e. $w_0^T = 1$ and $w_i^T = 0$ for all $i \neq 0$. This will force the chooser to move up $1/2 + \gamma$ of the chips in bin 0.

Following this line of reasoning, [Fre95] shows that the following backwards recursive weighting strategy is min-max optimal for the weightor

$$w_i^t = \left(\frac{1}{2} + \gamma\right)w_{i+1}^{t+1} + \left(\frac{1}{2} - \gamma\right)w_{i-1}^{t+1} \quad (3.2)$$

with final weighting on iteration T as $w_i^T = \mathbf{1}(i = 0)$. This recursion can be solved to give an explicit binomial equation for the weights

$$w_i^t = \begin{cases} \left(\frac{T-t}{2}\right) \left(\frac{1}{2} + \gamma\right)^{\frac{T-t-i}{2}} \left(\frac{1}{2} - \gamma\right)^{\frac{T-t+i}{2}} & \text{if } |i| \leq T-t \text{ and } (i-t) \text{ is odd} \\ 0 & \text{otherwise} \end{cases} \quad (3.3)$$

Similarly, a potential can be defined that corresponds to the *potential loss* associated with the chips in bin i on round t . At round $T + 1$, after the game has completed, any chips in bin $i < 0$ receive loss 1. Following this reasoning, the potential is defined as follows

$$\Phi_i^{T+1} = \begin{cases} 1 & \text{if } i < 0 \\ 0 & \text{otherwise} \end{cases} \quad (3.4)$$

$$\Phi_i^t = \left(\frac{1}{2} + \gamma\right)\Phi_{i+1}^{t+1} + \left(\frac{1}{2} - \gamma\right)\Phi_{i-1}^{t+1} \quad (3.5)$$

It is easy to verify that there is a simple relationship between the potential function defined this way and the min-max weighting scheme

$$w_i^t = \Phi_{i-1}^{t+1} - \Phi_{i+1}^{t+1} \quad (3.6)$$

¹If T is even then we have to make an arbitrary choice as to whether or not to include n_0^{T+1} in the sum and this choice complicates the notation without adding anything substantial to the analysis.

Moreover, using this weighting scheme, it is straightforward to show that a weak learner (chooser) that can produce weak rules with weighted error smaller than $\frac{1}{2} - \gamma$, and setting $T \geq \frac{1}{2\gamma^2} \ln \frac{1}{2\varepsilon}$ will produce a final combined rule whose training error is at most ε .

BrownBoost

The BBM algorithm takes a parameter T as input. As the number of boosting rounds approaches T , those examples with large positive or negative majority vote are given explicit weight zero. In other words, the algorithm gives up on these examples in the next iteration. This is an attractive behavior when label noise is introduced as we would hope that they would have such large scores and thus be given weight zero. However, the BBM algorithm is not a practical algorithm. This is because, it is not *adaptive* to the accuracies of the individual weak rules. Instead, the BBM algorithm performs an *unweighted* majority vote. In addition, the BBM algorithm requires knowledge of the advantage of each weak rule, γ , a difficult thing to measure in practice.

An adaptive version of the BBM algorithm is called BrownBoost and was derived in [Fre01]. Fundamentally, BB is the BBM game taken to the continuous time limit: a stochastic differential equation. We explain the derivation of BrownBoost here since it will help understand the reasoning behind RobustBoost.

We would like an algorithm that can use any rule whose error is smaller than $\frac{1}{2}$. Moreover, we'd like the final combined rule to be a *weighted* majority where rules with small error are assigned more weight than rules with large error. Consider a simple alteration to the BBM algorithm to achieve these goals, but without regard to computational complexity. Consider running the BBM algorithm but using a very small γ which will allow for rules that are only slightly better than random guessing. If we set γ to be extremely small, then the changes in the weights assigned to examples on consecutive iterations are also all extremely small. As a result, if the weighted error of the first rule h_1 is significantly better than $\frac{1}{2} - \gamma$, say $\frac{1}{4}$, then the weighted error of h_1 will continue to be better than $\frac{1}{2} - \gamma$ on subsequent iterations $2, 3, \dots$. After some number of iterations k_1 , the weighted error of h_1 with respect to the current weights will be larger than $\frac{1}{2} - \gamma$, at which point h_1 can no longer be used. At that point a new rule must be found h_2 which will “survive” until iteration $k_1 + k_2$, etc. The result is that the final majority vote will

have h_1 appearing k_1 times, h_2 appearing k_2 times, etc. In other words, we could represent the final rule as a *weighted majority* over the rules h_1, h_2, \dots with corresponding weights k_1, k_2, \dots .

Thus, in principle, we could use BBM as an adaptive boosting algorithm. Unfortunately, this straightforward implementation is prohibitively expensive to calculate in terms of computational time. We would like to be able to calculate k_i in an amount of time that does not depend on the size of k_i . BrownBoost formulates this problem as the solution of two nonlinear equations in two unknowns, allowing for the application of standard numerical techniques.

First, let us rewrite Equation (3.5) in a slightly different equivalent form.

$$\Phi_i^t - \Phi_i^{t-1} = (\Phi_i^t - \frac{1}{2}\Phi_{i+1}^t - \frac{1}{2}\Phi_{i-1}^t) - \gamma(\Phi_{i+1}^t - \Phi_{i-1}^t) \quad (3.7)$$

We now transform this equation to the continuous domain, replacing Φ_i^t defined over the integers with a new potential function defined over the reals $\Phi(s, t)$, where the real valued s takes the place of integer i . Also, we generalize the time step from the BBM algorithm from 1 to Δt and the location step from 1 to Δs . This new notation transforms Equation (3.7) into the following:

$$\begin{aligned} \Phi(s, t) - \Phi(s, t - \Delta t) &= -\frac{1}{2}(\Phi(s + \Delta s, t) - 2\Phi(s, t) + \Phi(s - \Delta s, t)) - \\ &\quad \gamma(\Phi(s + \Delta s, t) - \Phi(s - \Delta s, t)) \end{aligned} \quad (3.8)$$

We then let γ decrease towards zero while increasing the number of steps in the game towards infinity. From the BBM analysis we know that we need $T = \frac{\beta}{\gamma^2}$ iterations to achieve error ε with $\beta = \frac{1}{2} \ln \frac{1}{2\varepsilon}$. If we set the starting time for the game to be 0 and the final time to be 1 we get that $\Delta t = \frac{1}{T} = \frac{\gamma^2}{\beta}$. In addition, if we set $\Delta s = \sqrt{\Delta t} = \frac{\gamma}{\sqrt{\beta}}$ and divide both sides of Equation (3.8) by Δt we arrive at the following difference equation

$$\begin{aligned} \frac{\Phi(s, t) - \Phi(s, t - \Delta t)}{\Delta t} &= -\frac{1}{2} \frac{\Phi(s + \Delta s, t) - 2\Phi(s, t) + \Phi(s - \Delta s, t)}{\Delta s^2} \\ &\quad - \sqrt{\beta} \frac{\Phi(s + \Delta s, t) - \Phi(s - \Delta s, t)}{\Delta s} \end{aligned} \quad (3.9)$$

Taking the limit of the above as γ decreases to zero gives the following partial differential equation that describes the time evolution of a Brownian process

$$\frac{\partial \Phi(s, t)}{\partial t} = -\frac{1}{2} \frac{\partial^2 \Phi(s, t)}{\partial s^2} - 2\sqrt{\beta} \frac{\partial \Phi(s, t)}{\partial s} \quad (3.10)$$

Adapting the boundary condition for the end of the game to continuous time gives

$$\forall s, \quad \Phi(s, 1) = \begin{cases} 1 & \text{if } s < 0 \\ 0 & \text{otherwise} \end{cases}. \quad (3.11)$$

The above PDE can be solved in closed form for all $t \leq 1$:

$$\Phi(s, t) = \frac{1}{2} \left(1 - \operatorname{erf} \left(\frac{s + 2\sqrt{\beta}(1-t)}{\sqrt{2(1-t)}} \right) \right) \quad (3.12)$$

where erf is the Gaussian error function. We can also express the weighting function as follows:

$$w(s, t) = \frac{\partial}{\partial s} \Phi(s, t) = \exp \left(-\frac{(s + 2\sqrt{\beta}(1-t))^2}{2(1-t)} \right) \quad (3.13)$$

Finally, we set β according to the value of ε , the target training error which we wish to achieve. Specifically, we choose β to satisfy the equation

$$\varepsilon = \Phi(0, 0) = \frac{1 - \operatorname{erf}(\sqrt{2\beta})}{2} \quad (3.14)$$

What remains to be discussed to complete BrownBoost is how to update t , the time variable, and $s(j)$ the position of example j on the real line. It is shown in [Fre01] that these updates are dictated by the following two equations in two unknowns $(\Delta t_k, \Delta s_k)$:

$$\sum_{j=1}^m y_j h_k(x_j) w(s(j) + y_j h_k(x_j) \Delta s_k, t_k + \Delta t_k) = 0 \quad (3.15)$$

$$\sum_{j=1}^m \Phi(s(j), t_k) = \sum_{j=1}^m \Phi(s(j) + y_j h_k(x_j) \Delta s_k, t_k + \Delta t_k) \quad (3.16)$$

Here the first equation says that our current weak rule should have no advantage on the weighted training set in the next iteration. The second equation says that the average potential should not change, which is a requirement to a solution to the PDE above, as shown in Theorem 2 of [Fre01]. Using a numerical technique like Newton-Raphson, we can solve the above system of equations. On iteration k the time is updated with $t_{k+1} = t_k + \Delta t$. The algorithm halts when $t_{k+1} \geq 1$ leaving us with the score distribution across training examples given by $s(j)$. BrownBoost takes only one parameter as input,

the target error ε . If the algorithm halts, we know that no more than ε fraction of our training set has $s(j)$ below zero.

BrownBoost is an adaptive boosting algorithm that tries to optimize the number of classification mistakes on the training set. This goal is captured by setting the final potential at the end of the game, defined in Equation (3.11), to be the step function with step placed at $s = 0$. However, the large margin theory of boosting described by Equation (3.1) gives a view of boosting beyond that of just minimizing the training error, namely we should be trying to achieve large margins instead.

Suppose we augmented the final potential function of BrownBoost to be a step function centered at $s = 1$ instead of $s = 0$. This will indeed put pressure on creating larger margins, but not in a very meaningful way. To achieve a similar effect, we could have taken the output of the $s = 0$ BrownBoost and scaled each α_i up so that all the correct training examples as a result have larger margins. With this trick we can create arbitrary sized margins for correctly classified examples without fundamentally changing the margin distribution. It's clear from this thought experiment that simply maximizing unnormalized margins is not what we want, and the large margin theorem of Equation (3.1) agrees. It says we should instead be looking at the ℓ_1 -normalized margin

$$\frac{y \sum_i \alpha_i h_i(x)}{\sum_i |\alpha_i|}$$

RobustBoost

To address the problem of BrownBoost maximizing unnormalized margins, Freund suggested another algorithm called RobustBoost in [Fre09]. RobustBoost is designed to limit the *variance* of the margin distribution

$$\frac{y \sum_i \alpha_i h_i(x)}{\text{Var}(\sum_i \alpha_i h_i(x))}$$

Note there is a similarity here to the margin bounds shown in the SVM literature with typically have an ℓ_2 -normalization. RobustBoost takes two key parameters as input: a target error ε and a final margin θ . RobustBoost simultaneously limits the variance of the score distribution to a constant and attempts to get $1 - \varepsilon$ fraction of the training examples above a margin of θ .

The derivation of RobustBoost starts by altering the evolution of the continuous time version of the BBM game. In BrownBoost examples move from (s, t) to $(s \pm \Delta s, t + \Delta t)$. For RobustBoost we instead move examples from (s, t) to $(s(1 - \Delta t) \pm \Delta s, t + \Delta t)$. This adds a drift in the underlying Brownian motion which pushes examples towards zero and limits the variance of the score distribution. Adding this drift changes Equation (3.9) into

$$\frac{\Phi(s, t) - \Phi(s, t - \Delta t)}{\Delta t} = -\frac{1}{2} \frac{\Phi(s(1 - \Delta t) + \Delta s, t) - 2\Phi(s, t) + \Phi(s(1 - \Delta t) - \Delta s, t)}{\Delta s^2} - \sqrt{\beta} \frac{\Phi(s(1 - \Delta t) + \Delta s, t) - \Phi(s(1 - \Delta t) - \Delta s, t)}{\Delta s}$$

Set $\rho = \sqrt{\beta}$ and $\Delta s^2 = \Delta t = \frac{\gamma^2}{\beta}$ like in BrownBoost. After rearranging we get

$$\begin{aligned} & \frac{\Phi(s, t) - \Phi(s, t - \Delta t)}{\Delta t} \\ &= -\frac{1}{2} \frac{\Phi(s(1 - \Delta s^2) + \Delta s, t) - 2\Phi(s(1 - \Delta s^2) + \Delta s, t) + \Phi(s(1 - \Delta s^2) - \Delta s, t)}{\Delta s^2} \\ & \quad - \rho \frac{\Phi(s(1 - \Delta s^2) + \Delta s, t) - \Phi(s(1 - \Delta s^2) - \Delta s, t)}{\Delta s} \\ & \quad + \frac{\Phi(s, t) - \Phi(s(1 - \Delta s^2), t)}{\Delta s^2} \end{aligned} \tag{3.17}$$

We now let $\gamma \rightarrow 0$ as in BrownBoost giving the following differential equation

Claim 3.1.

$$\lim_{h \rightarrow 0} \frac{f(x) - f(x(1 - h))}{h} = x \frac{df}{dx}$$

Proof. By Taylor expansion of $f(x(1 - h))$ around $f(x)$ we arrive at

$$\lim_{h \rightarrow 0} \frac{f(x) - [f(x) - xhf'(x) + h^2\mathcal{R}(x, h)]}{h} \tag{3.18}$$

which gives the desired result. \square

$$\frac{\partial \Phi(s, t)}{\partial t} = -\frac{1}{2} \frac{\partial^2 \Phi(s, t)}{\partial s^2} + (s - 2\rho) \frac{\partial \Phi(s, t)}{\partial s} \tag{3.19}$$

This follows from the limit definition of a partial derivative. The term $s \frac{\partial \Phi(s, t)}{\partial s}$ can be seen after observing a fact from elementary calculus $\lim_{h \rightarrow 0} \frac{f(x) - f(x(1 - h))}{h} = x \frac{df}{dx}$ (shown

by using a Taylor expansion of $f(x(1-h))$ around $f(x)$. The other terms can be derived in an identical manner.

The PDE in (3.19) corresponds to the backwards Kolmogorov equation for the mean-reverting Ornstein-Uhlenbeck diffusion process. Intuitively, this process has a constant pressure placed on the current value to revert back towards the mean.

It can be easily verified that the following is a family of solutions for the PDE in Equation (3.19)

$$\Phi(s,t) = \frac{1}{2} \left(1 - \operatorname{erf} \left(\frac{s - \mu(t)}{\sigma(t)} \right) \right) \quad (3.20)$$

Where

$$\sigma(t) = \sqrt{c_1 e^{-2t} - 1}$$

$$\mu(t) = c_2 e^{-t} + 2\rho$$

and c_1, c_2 are real valued constants. We set these constants so that the final potential function is such that $\mu(1) = \theta$ and $\sigma(1) = \sigma_f$. These will be two parameters of the RobustBoost algorithm, namely the final margin to shoot for θ and something similar to the standard-deviation of the final potential σ_f . Solving for these constraints we get

$$\sigma(t) = \sqrt{(\sigma_f^2 + 1)e^{2(1-t)} - 1} \quad (3.21)$$

$$\mu(t) = (\theta - 2\rho)e^{1-t} + 2\rho \quad (3.22)$$

We set the value of ρ according to our target loss ε . Specifically we find ρ that satisfies

$$\varepsilon = \Phi(0,0) = \frac{1}{2} \left(1 - \operatorname{erf} \left(\frac{2(e-1)\rho - e\theta}{\sqrt{e^2(\sigma_f^2 + 1) - 1}} \right) \right) \quad (3.23)$$

We can also define a weight function similar to the logic used in BrownBoost as follows

$$w(s,t) = \frac{\partial}{\partial s} \Phi(s,t) = \begin{cases} \exp \left(-\frac{(s-\mu(t))^2}{2\sigma(t)^2} \right) & \text{if } s > \mu(t) \\ 0 & \text{if } s \leq \mu(t) \end{cases} \quad (3.24)$$

We are now ready to present the RobustBoost algorithm given in Figure 3.3. Note that the algorithm terminates when t_{k+1} is exactly 1.

Like in BrownBoost, a system of equations in two unknowns (3.15-3.16) must be solved with the addition of each weak hypothesis, however the score updates in RobustBoost are slightly different. Remember that in BrownBoost the score position updates are $s(j) := s(j) + y_j h_k(x_j) \Delta s_k$, whereas in the algorithm for RobustBoost there is a damping factor

$$s(j) := s(j)e^{-\Delta t_k} + y_j h_k(x_j) \Delta s_k \quad (3.25)$$

The justification is as follows. Suppose that we divide the time step Δt into n equal parts of length $\Delta t/n$. At each of these steps $i = 1, \dots, n$ we solve for Δs_i and update s , assuming that Δt is sufficiently small, we know that $|\Delta s_i| \leq \sqrt{\Delta t/n}$. The step from s_0 to s_1 .

$$s_1 = s_0(1 - \Delta t/n) + \Delta s_0$$

if we expand this formula to two steps we get

$$\begin{aligned} s_2 &= (s_0(1 - \Delta t/n) + \Delta s_0)(1 - \Delta t/n) + \Delta s_1 \\ &= s_0(1 - \Delta t/n)^2 + \Delta s_0 - \Delta s_0 \Delta t/n + \Delta s_1 \end{aligned}$$

recalling that $|\Delta s_0| \leq \sqrt{\Delta t/n}$, this means that the term $\Delta s_0 \Delta t/n$ is smaller than $(\Delta t)^{3/2}$ thus as $n \rightarrow \infty$ this term becomes negligible compared to Δs_0 and Δs_1 . In other words

$$s_2 = s_0(1 - \Delta t/n)^2 + \Delta s_0 + \Delta s_1 + O(n^{-3/2})$$

We can now recurse over all n steps and get

$$s_n = s_0(1 - \Delta t/n)^n + \sum_{i=1}^n \Delta s_i + O(n^{-1/2})$$

Taking the limit $n \rightarrow \infty$ we get Equation (3.25).

3.3 Adaptive- ε Heuristic

In BrownBoost and RobustBoost, we need to specify the target error rate ε . The choice of ε can greatly influence the performance of the trained classifier. When ε is set too low or too high, the algorithms often produce a classifier that performs poorly even on the training data. Figure 3.4 shows the margin distributions of BrownBoost and

Given: $\varepsilon > 0$, $\theta > 0$, $\sigma_f > 0$

$(x_1, y_1), \dots, (x_m, y_m)$ where $x_j \in \mathcal{X}$, $y_j \in \mathcal{Y} = \{-1, +1\}$

Set ρ to satisfy Equation (3.23).

Initialize $t_1 = 0$, $H_0 \equiv 0$ and $s(j) := 0$ for all $1 \leq j \leq m$.

Repeat for $k = 1, 2, \dots$

- Define the distribution D_k over the m training examples by normalizing $w(s, t)$ defined in Equation (3.24)

$$D_k(j) = \frac{w(s(j), t_k)}{Z}, \quad Z = \sum_{j=1}^m w(s(j), t_k)$$

- Get weak hypothesis $h_k : \mathcal{X} \rightarrow \{-1, +1\}$ which is slightly correlated with the label:

$$\mathbb{E}_{j \sim D_k} [y_j h_k(x_j)] > 0$$

- **Find** $\Delta s_k > 0$, $1 - t_k \geq \Delta t_k > 0$ that simultaneously satisfy the following two equations:

$$\sum_{j=1}^m y_j h_k(x_j) w(s'(j), t_k + \Delta t_k) = 0 \quad \text{or} \quad \Delta t_k = 1 - t_k$$

$$\sum_{j=1}^m \Phi(s(j), t_k) = \sum_{j=1}^m \Phi(s'(j), t_k + \Delta t_k)$$

where

$$s'(j) \doteq s(j)e^{-\Delta t_k} + y_j h_k(x_j) \Delta s_k$$

and $\Phi(\cdot, \cdot)$, $w(\cdot, \cdot)$ are defined by Equations (3.20, 3.24)

- **update:** $t_{k+1} := t_k + \Delta t_k$, $\forall 1 \leq j \leq m$, $s(j) := s'(j)$

$$H_k = H_{k-1} e^{-\Delta t_k} + \Delta s_k h_k$$

- **break** if $t_{k+1} = 1$.

Output the final hypothesis H_k .

Figure 3.3: RobustBoost algorithm.

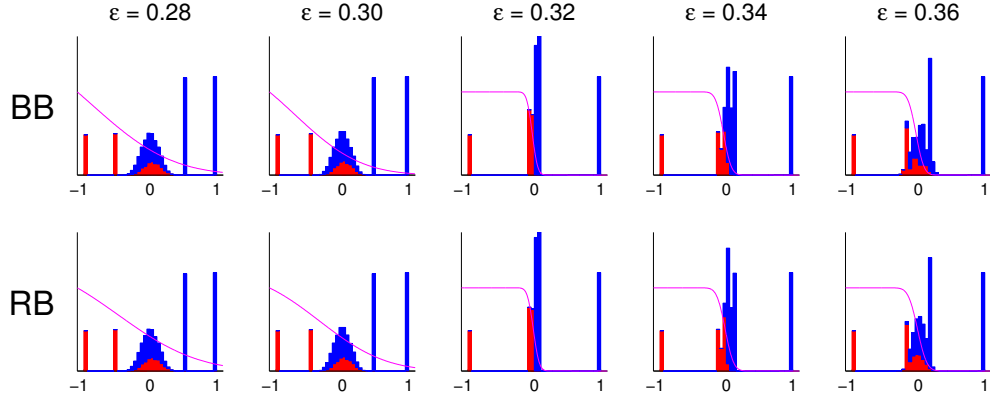


Figure 3.4: Margin distributions of BrownBoost (BB) and RobustBoost (RB) using different ε on LS dataset with 30% label noise after 200 iterations. Noisy and clean examples are shown in red and blue respectively.

Table 3.1: Various training parameters of BrownBoost (BB) and RobustBoost (RB) using ε slightly below and above the noise level η . The final time and the final training error rate with respect to clean labels are denoted by t_f and E_f respectively. The angle between the true hypothesis \vec{h} and $\vec{\alpha}$ of BB and RB using ε slightly below and above the noise level η is shown in the last row.

		$\eta = 0.10$		$\eta = 0.20$		$\eta = 0.30$	
		$\varepsilon = \eta - 0.02$	$\varepsilon = \eta + 0.02$	$\varepsilon = \eta - 0.02$	$\varepsilon = \eta + 0.02$	$\varepsilon = \eta - 0.02$	$\varepsilon = \eta + 0.02$
t_f	BB	0.19	0.30	0.22	0.81	0.25	0.82
	RB	0.21	0.57	0.25	0.78	0.27	0.81
E_f	BB	0.40	0.14	0.34	0.00	0.28	0.00
	RB	0.40	0.00	0.34	0.00	0.28	0.00
$\frac{\vec{\alpha} \cdot \vec{h}}{\ \vec{\alpha}\ \cdot \ \vec{h}\ }$	BB	0.51	0.30	0.48	0.09	0.53	0.00
	RB	0.51	0.11	0.48	0.08	0.53	0.00

RobustBoost using different ε on a dataset with 30% label noise after 200 iterations. Note that when $\varepsilon \leq 0.30$, the classifier cannot separate examples around zero-margin.

When the true noise rate is known, a good rule of thumb is to set ε a little higher than the noise rate. Table 3.1 summarizes the final time t_f and the final training error rate E_f of BrownBoost and RobustBoost on a synthetic dataset with the true noise rate $\eta = \{0.1, 0.2, 0.3\}$ using two different ε settings slightly above and below η . However, when the true noise rate is not known, the tuning of ε is usually done by cross validation. The process can be very inefficient and usually involves a grid search over a small interval.

We propose a heuristic for tuning ε automatically for BrownBoost and RobustBoost. The idea is based on the following observation. When ε is too small, the time t_k advances too slowly that the boosting procedure seems “stuck”. This situation can

Algorithm 3 RobustBoost/BrownBoost with adaptive- ε heuristic

Initial Assumptions: The maximum number of boosting iterations T .

```

1: Initialize  $\varepsilon \leftarrow 0$  and  $tries \leftarrow 0$ .
2: for  $iter \leftarrow 1$  to  $T$  do
3:   Solve for  $\Delta s$  and  $\Delta t$ .
4:   if  $\Delta t$  is too small or the solver fails to converge then
5:      $tries \leftarrow tries + 1$ 
6:     if  $tries > max\_tries$  then
7:       Increase  $\varepsilon$  by a small amount.
8:        $tries \leftarrow 0$ 
9:     end if
10:  else
11:     $tries \leftarrow 0$ 
12:  end if
13: end for

```

often be remedied by slightly increasing the value of ε without having to restart the boosting process. The heuristic can be described as follows. Initially, we set $\varepsilon = 0$ and start the boosting procedure. When the boosting algorithm does not to advance for a few iterations or the numerical solver fails to solve the non-linear equations, we slightly increase ε and resume the boosting process. In our experiments, we denote BrownBoost and Robust with adaptive- ε heuristic and RobustBoost with adaptive- ε , BBA and RBA respectively. The algorithms are summarized in Algorithm 3.

3.4 Experiments

In this section we compare the performance of BrownBoost with adaptive- ε (BBA), RobustBoost with adaptive- ε (RBA) to AdaBoost (ADB) and LogLossBoost (LLB)² on 3 datasets with and without random label noise. Specifically we will look closely at the margin distributions for each of the boosting algorithms. Additionally, we will study the impact of using positive target margin θ on BBA and RBA.

²LogLossBoost is our implementation of LogitBoost with decision stumps.

We implemented all of the boosting algorithms in MATLAB and utilized the Optimization Toolbox for numerically solving BB and RB equations. For RBA, we use $\sigma_f = 0.001$ for all experiments.

Datasets

We conducted experiments on 3 different datasets: LS, Face and Satimage. Each dataset can be described as follows. First, LS dataset is a synthetic dataset whose construction is suggested by Long and Servidio in [LS08]. The dataset has input $x \in \mathbb{R}^{21}$ with binary features $x_i \in -1, +1$ and label $y \in -1, +1$. Each instance is generated as follows. First, the label y is chosen to be -1 or $+1$ with equal probability. Given y and the margin width parameter δ , the features x_i are chosen according to the following mixture distribution:

- **Large margin:** With probability $1/4$, we choose $x_i = y$ for all $1 \leq i \leq 21$
- **Pullers:** With probability $1/4$, we choose $x_i = y$ for $1 \leq i \leq 10 + \delta$ and $x_i = -y$ for $11 + \delta \leq i \leq 21$
- **Penalizers:** With probability $1/2$, we choose $5 + \lfloor \delta/2 \rfloor$ random coordinates from the first 11 and $5 + \lceil \delta/2 \rceil$ from the last 10 to be equal to the label y . The remaining 10 coordinates are equal to $-y$.

The data from this distribution can be classified perfectly by a simple linear classifier $f(x) = \text{sgn}(\sum_i x_i)$. Note that δ essentially controls the separation margin of the examples. Larger δ yields a larger margin.

Face dataset is a collection of face and non-face images consisting of 10000 face images and 20000 non-faces images. For each image, a feature vector of \mathbb{R}^{176} is calculated based on histogram of colors and gradients. When label noise is added to Face dataset, we only added noise to the negative examples. We use 70% of the examples for training and 20% for testing.

Satimage is a dataset from the UCI repository [BL13]. There are 6435 examples and 36 attributes. The original label of 1-3 is grouped as $+1$ and the rest is group as -1 . Similar to Face dataset, we use 70% of the examples for training and 20% for testing.

Table 3.2: Average test error rates of ADB, LLB, BBA and RBA with respect to the noisy labels (n) and the true labels (t) on LS dataset with $N=1600$ in different noise settings.

LS w/ $\delta = 1$		ADB	LLB	BBA	RBA
$\eta = 0.0$	n	-	-	-	-
	t	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
$\eta = 0.1$	n	0.25 (0.01)	0.24 (0.01)	0.10 (0.01)	0.10 (0.01)
	t	0.23 (0.01)	0.22 (0.01)	0.00 (0.00)	0.01 (0.01)
$\eta = 0.2$	n	0.32 (0.01)	0.31 (0.01)	0.21 (0.01)	0.22 (0.02)
	t	0.23 (0.01)	0.23 (0.01)	0.03 (0.02)	0.05 (0.03)
$\eta = 0.3$	n	0.36 (0.01)	0.36 (0.01)	0.31 (0.02)	0.32 (0.02)
	t	0.24 (0.01)	0.24 (0.01)	0.09 (0.05)	0.12 (0.05)

LS w/ $\delta = 3$		ADB	LLB	BBA	RBA
$\eta = 0.0$	n	-	-	-	-
	t	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
$\eta = 0.1$	n	0.11 (0.01)	0.10 (0.01)	0.10 (0.01)	0.10 (0.01)
	t	0.02 (0.01)	0.00 (0.00)	0.01 (0.00)	0.00 (0.00)
$\eta = 0.2$	n	0.22 (0.01)	0.21 (0.01)	0.19 (0.01)	0.19 (0.01)
	t	0.06 (0.02)	0.04 (0.02)	0.02 (0.00)	0.01 (0.01)
$\eta = 0.3$	n	0.33 (0.01)	0.33 (0.01)	0.29 (0.01)	0.29 (0.01)
	t	0.12 (0.02)	0.11 (0.02)	0.04 (0.01)	0.03 (0.01)

Results

We first compared the performance of ADB, LLB, BBA and RBA with different label noise level $\eta \in 0.0, 0.1, 0.2, 0.3$ on LS using 2 settings of the margin width parameter $\delta \in 1, 3$. We used the training set size $N = 1600$ and ran each boosting algorithm for 200 iterations. For BBA and RBA, the margin parameter $\theta = 0$ and $\sigma_f = 0.001$.

When there was no label noise, all boosting algorithms managed to learn the correct linear classifier. However, with presence of label noise in both settings of δ , BBA and RBA successfully converged to the correct classifier while ADA and LLB did not as indicated by the higher test error rates with respect to the true labels. Table 3.2 summarizes the average test error rates with respect to the true labels and the noisy labels over 10 runs and the standard deviation is reported in parentheses.

We also examined the progression of the margin distributions for each of the boosting algorithms. Figure 3.7 shows the margin distributions progression when $\eta = 0.3$. In LS with $\delta = 1$, ADB and LLB stopped progressing after 50 iterations due to the

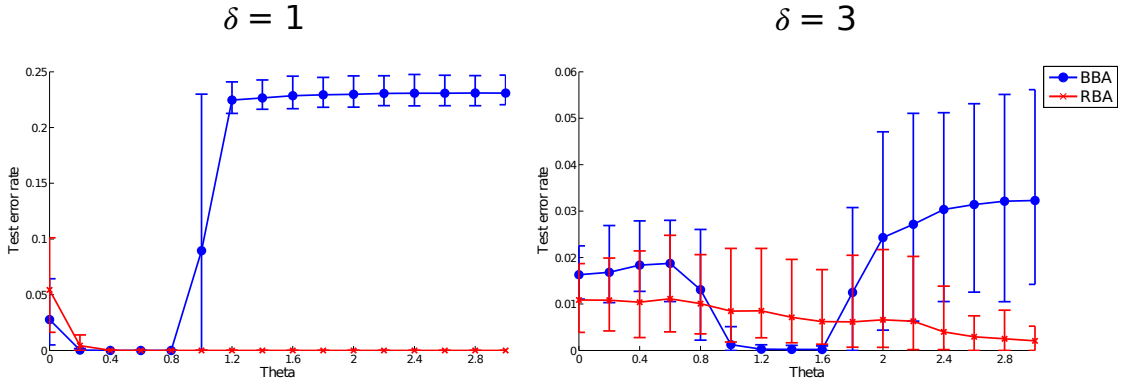


Figure 3.5: Test errors of RBA and BBA using different θ on LS with 20% noise. The whiskers indicate the minimum and maximum values over 10 runs.

large weights put on the noisy large-margin examples pushing the classifier away from the correct hypothesis. On the contrary, after 100 iterations, RBA and BBA significantly decreased the weights of the noisy large-margin examples as these examples are being “given up”. As a result, the boosting process continued on and eventually converged to the correct classifier. Interestingly, in LS with $\delta = 3$, we found that all boosting algorithms managed to attain the training error rate of 0 in all noise settings. However, the test error rates of ADB and LLB remained relatively high compared to those of BBA and RBA.

We further explored the benefits of the margin parameter θ . We found that using a positive θ can improve generalization error. Figure 3.5 shows the test errors of BBA and RBA using different θ on LS with 20% noise. Both algorithms have lower generalization error when using positive θ . We found that RBA is less sensitive to the setting of θ than BBA. Figure 3.6 summarizes the test error rates as a function of training set size for LS dataset with $\delta = 1$ and $\delta = 3$. For BBA and RBA, θ is tuned by cross-validation.

We also compared the performance of ADB, LLB, BBA and RBA on Face and Satimage. We ran each boosting algorithm for 800 iterations on both datasets with 2 different noise levels $\eta = 0.0, 0.2$. We also repeated the experiment after holding out of 75% of the training examples. The area under the average ROC curves is summarized in Table 3.3.

For both Face and Satimage, using paired t-test, we found that area under ROC of RBA and BBA is significantly larger than that of LLB with $p < 0.001$ in all settings.

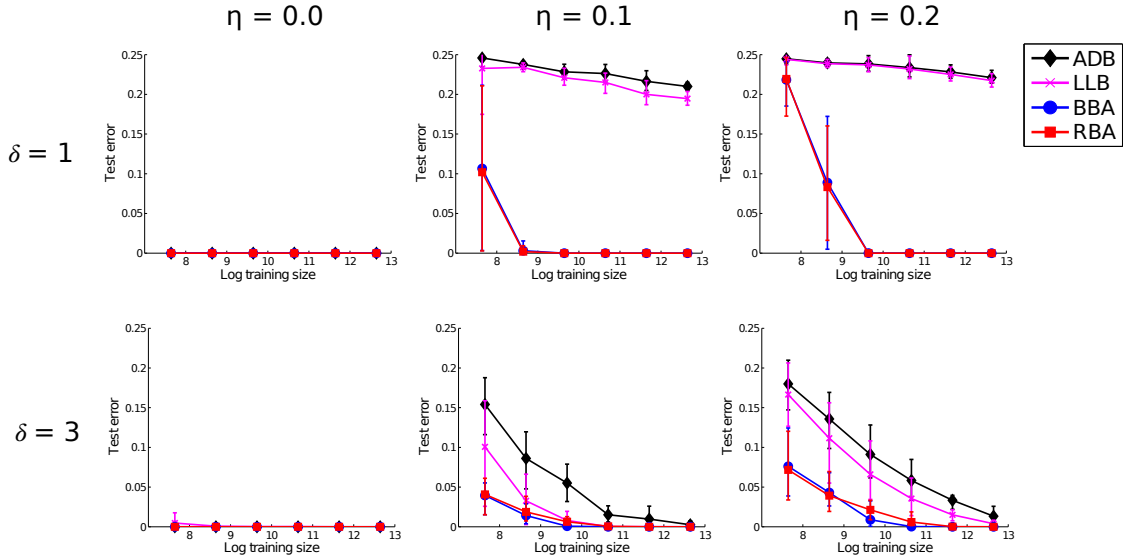


Figure 3.6: Test errors of ADB, LLB, BBA, RBA while varying number of training examples at different noise levels η . The whiskers indicate the minimum and maximum values over 10 runs.

We also found that the difference between ADB and RBA is insignificant in noise-free cases and the difference between RBA and BBA is insignificant in all cases. For Satimage dataset,

3.5 Conclusions

Our experiments show that Brownboost and Robustboost are significantly more resistant to label noise than Adaboost and LogitBoost. We show how this is related to the progressions of the margin distribution over time. We show that the setting of the target error rate ε is of critical importance for the final performance and provide a practical heuristics for setting it. Our experiments also show that, for noisy small training sets, maximizing the margin on the examples on which we don't “give up” is significantly better than minimizing the training error.

Table 3.3: Average area under ROC of ADB, LLB, BBA and RBA on Face and Satimage.

	ADB	LLB	BBA	RBA
$\eta = 0.0, N = 21000$	0.9996 (0.0001)	0.9979 (0.0004)	0.9996 (0.0001)	0.9996 (0.0000)
$\eta = 0.0, N = 5250$	0.9998 (0.0000)	0.9994 (0.0001)	0.9998 (0.0000)	0.9997 (0.0000)
$\eta = 0.2, N = 21000$	0.9991 (0.0001)	0.9992 (0.0001)	0.9995 (0.0001)	0.9995 (0.0001)
$\eta = 0.2, N = 5250$	0.9983 (0.0003)	0.9982 (0.0003)	0.9992 (0.0001)	0.9993 (0.0001)

(a) Face

	ADB	LLB	BBA	RBA
$\eta = 0.0, N = 4504$	0.9830 (0.0003)	0.9832 (0.0007)	0.9770 (0.0010)	0.9757 (0.0005)
$\eta = 0.0, N = 1126$	0.9764 (0.0018)	0.9720 (0.0021)	0.9770 (0.0016)	0.9729 (0.0022)
$\eta = 0.2, N = 4504$	0.9679 (0.0026)	0.9699 (0.0021)	0.9749 (0.0018)	0.9715 (0.0038)
$\eta = 0.2, N = 1126$	0.9459 (0.0043)	0.9421 (0.0053)	0.9607 (0.0039)	0.9656 (0.0047)

(b) Satimage

Acknowledgements

This chapter is based on unpublished work that is currently in submission as of the writing of this thesis. It is joint work with Evan Ettinger and Yoav Freund. The dissertation author is the primary investigator and author of this work.

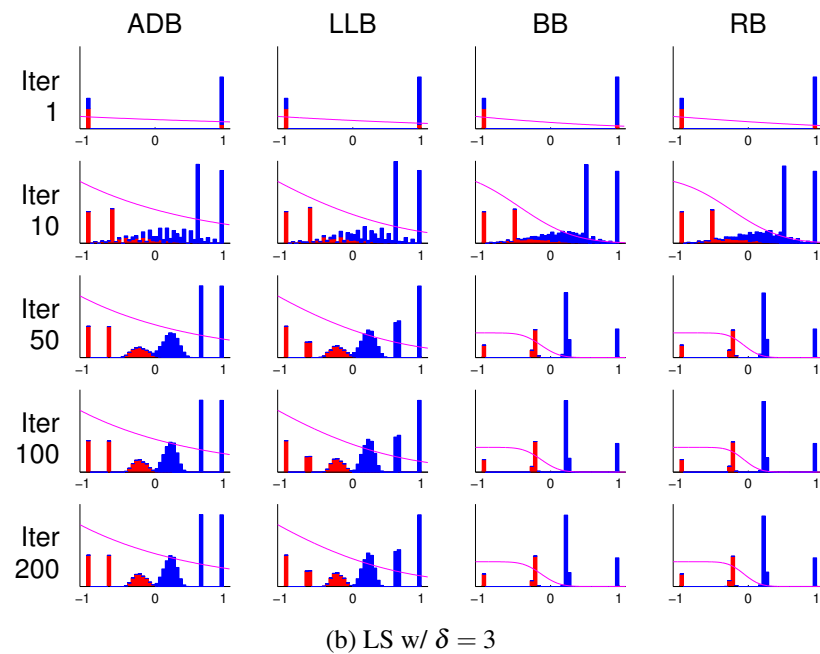
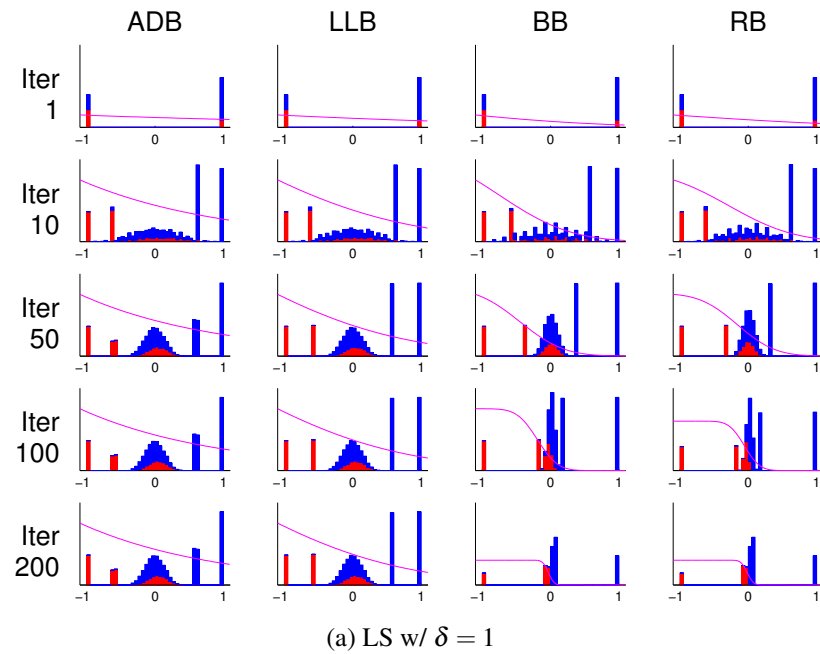


Figure 3.7: Margin distribution progression and potential loss function of ADB, LLB, BBA and RBA on LS with 30% label noise. Noisy and clean examples are shown in red and blue respectively.

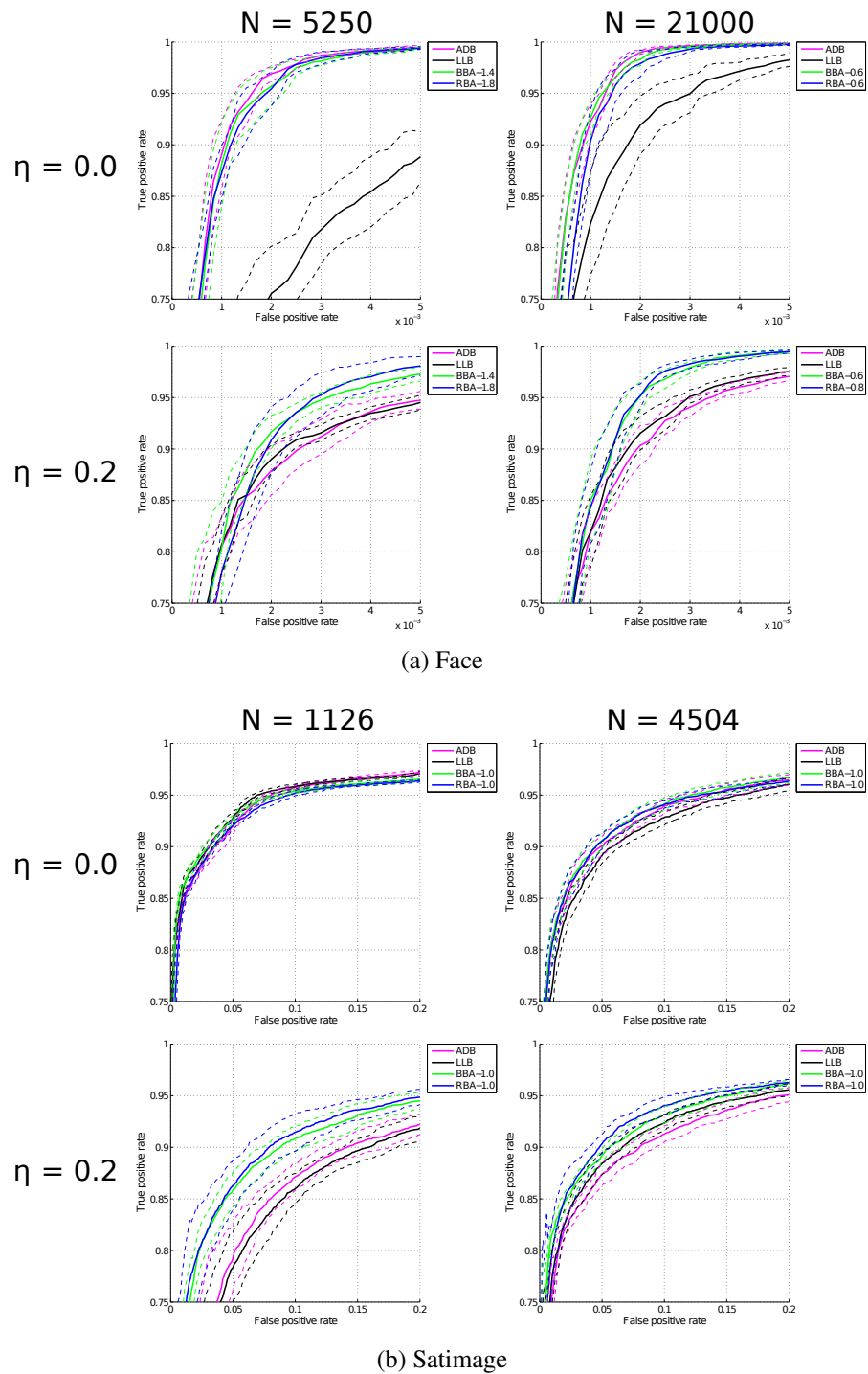


Figure 3.8: Average ROC curves of ADB, LLB, BBA and RBA while varying training size and noise level. The dotted lines correspond to the standard deviation. The chosen values of θ are included in the legend.

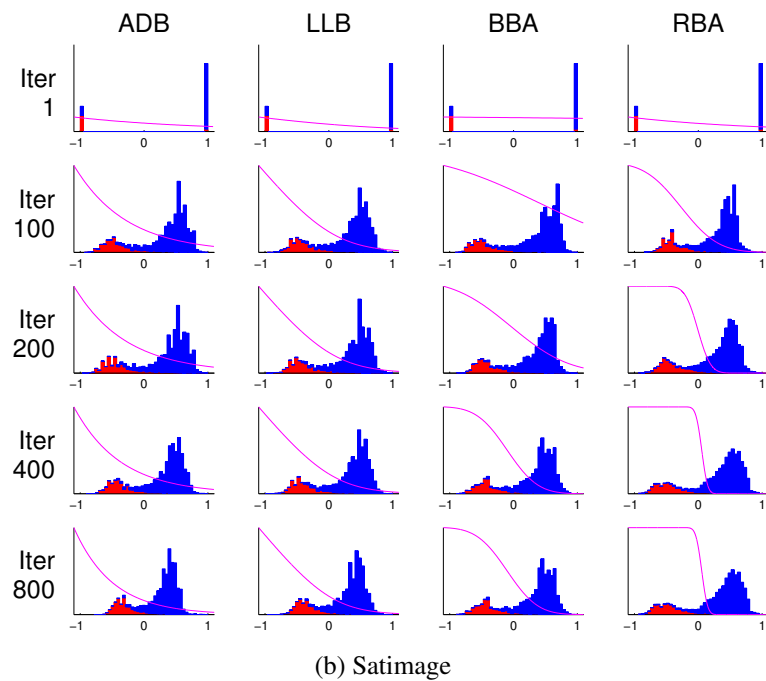
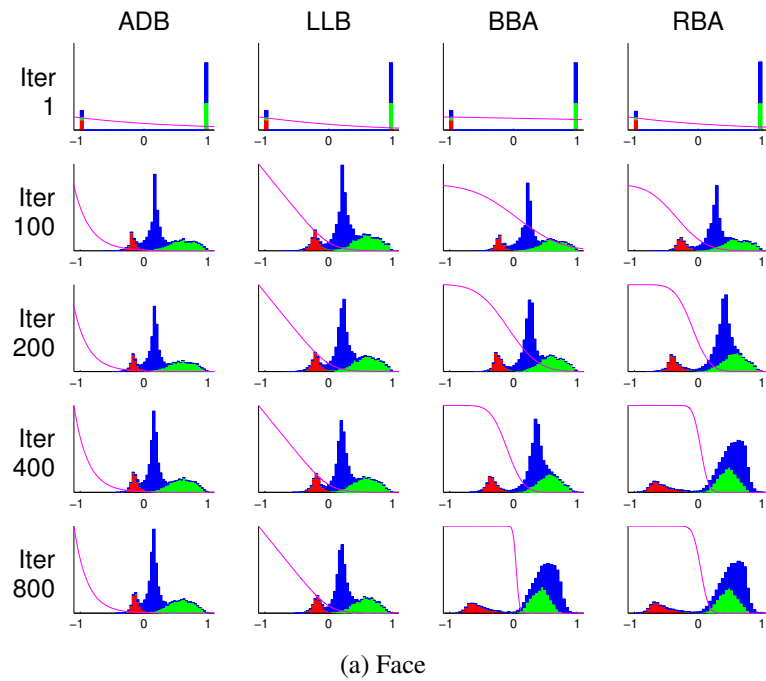


Figure 3.9: Margin distribution progression and potential loss function of ADB, LLB, BBA and RBA on Face and Satimage. For Face, true positive and fake positive examples are shown in green and red respectively. Negative examples are shown in blue. For Satimage, noisy and clean examples are indicated by red and blue respectively.

Chapter 4

uRight: Co-adaptive Handwriting Recognition System

Handwriting recognition is an input method that utilizes the screen space efficiently. With a small writing area on the screen, users can potentially compose a message using a mixed of characters from multiple languages, without having to switch between different keyboard layouts. Technically, the task of handwriting recognition is to translate a spatial representation of a character into its semantic meaning e.g. translate the ink trail of “a” into the letter “a” [PS00]. The difficulty of such task stems from the followings.

- *Between-writer variation.* Different people write differently even though their intents are the same. The difference can be cultural or context-dependent. This is demonstrated by the variation between rows in Figure 4.1. This problem can be addressed by training a recognizer for a particular writer.
- *Within-writer variation.* When a particular writer writes a specific letter several times, each instance would be slightly different. This is demonstrated in Figure 4.1. This effect increases with writing speed. While within-writer variation is usually smaller than between user variation, it is harder to solve.
- *Segmentation.* The handwriting recognition problem is inexorably entangled with handwriting *segmentation*. In order to recognize the individual characters, we

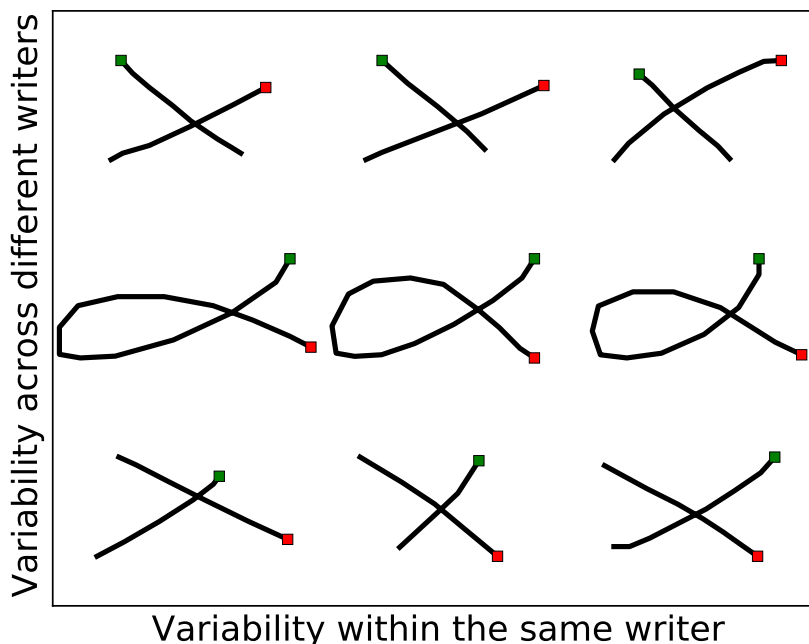


Figure 4.1: Handwritten character “x” from 3 different writers (vertically) and the variation of the handwriting within the same writer (horizontally). The green and red dots indicate where the writing starts and stops respectively.

need to first partition the handwriting into individual letters. On the other hand, finding the correct way to segment the handwriting requires identifying the individual characters. This is a difficult task, especially for cursive handwriting, and often is avoided by performing the recognition at the word level, or by requiring the user to write in a certain, restricted way e.g. using exactly one stroke for each character.

Machine learning methods are commonly used in handwriting recognition [PS00]. Moreover, It has long been recognized that, in order to get high classification accuracy, it is beneficial to train the classifier specifically for the particular user. There has been significant work on such personalization [MGDV93, CJ02]. However, all of the work on personalization that we are aware of is based on passive data collection. The user is asked to write a predefined set of sentences, independent of the properties of the user’s handwriting. In addition, there is no attempt to teach the user how to write in a way that would make recognition easier.

The need to balance computer adaptation and human adaptation is manifest in two of the early commercial handwriting recognition systems: Apple's Newton [YWL98] and PalmOS Graffiti. The Newton was one of the early handwriting recognition systems which was based on neural networks. It did adapt to the individual user by default. Unfortunately, the result was so poor that it was parodied by both Doonsbury and the Simpsons. Coming on the heels of the Newton's failure, the choice of the PalmOS developers was to adopt the Graffiti system which enforced a predefined set of character gestures to which the user had to adapt.

In this chapter we present an adaptive handwriting recognition system called *uRight* that we have built to investigate the adaptive capabilities of both the human and the machine. We first give an overview of the system in Section 4.1. Then, in Section 4.3, we discuss the choice of adaptive recognition algorithms that we have considered to include in the system.

4.1 Co-adaptive Handwriting Recognition System

Past research in adaptive handwriting recognition can be partitioned into two broad categories: *machine adaptation* and *human adaptation*. The *machine adaptation* approach places the responsibility for adaptation on the computer. In this approach machine learning algorithms are trained on past instances of handwriting to improve the future error rate. A number of machine learning algorithms have been applied to the problem such as time delay neural network [JMW00] and support vector machines [BHB02]. Generative models such as Hidden Markov model (HMM) are commonly used to model handwritten characters [TG11].

The *human adaptation* approach requires the human to adapt to the computer. This is usually done by defining a specific handwriting letter set and requiring the user to learn it. Unistrokes [GR93] and Palm's Graffiti are notable examples. Such systems require significant training on the part of the user, in exchange, they provide low error rate and high communication rate [MZ97].

In a sense, human adaptation is always present, whether or not the recognition algorithm takes advantage of it. Users engaging in a handwriting interface will, in some

cases, try to change the way in which they write to reduce errors and increase speed. In other cases they might dig in their heels and keep writing in the way they are used to.

We consider a new approach to the handwriting recognition, called *co-adaptation*, that combines *machine adaptation* with *human adaptation*. The key characteristics of the *co-adaptive* handwriting recognition system are the followings.

- Adapting to each individual user. The recognition algorithm re-estimates the handwriting model for each user using the past observations. This process is also referred to as personalization.
- Assisting the user to adapt to the system. The system constantly provides feedback to the users to teach/guide them to achieve a better performance.

Each property of the *co-adaptive* system is described in detail as follows.

Personalization

The concept of personalization, or *writer-dependent* recognition, has been well-established and shown to perform better than the writer-independent counterpart in term of recognition accuracy [MGDV93, CJ02]. The technique addresses the individual difference of handwriting by adapting to each user individually. For example, some people write faster than others, some people write smaller than others. By specializing the recognizer for each user, the recognition task becomes easier because the variability of the handwriting across different users is eliminated. Moreover, the personalization fits well with the mobile platform where there is only one primary user.

However, the lack of data in the initial sessions is a challenge for personalization. Obviously, some handwriting examples from each user is required in order to “train” the personalized recognizer. To overcome this problem, the system could ask the user to engage in a dedicated training session. We believe that a handwriting recognition system should work “out of the box”. Therefore, we decide not to have an explicit training session but, instead, we initialize the recognition algorithm with a recognizer that was trained using data from multiple users, *writer-independent* recognizer, and then gradually tune it according to the handwriting of each user as more data become available. We will discuss the recognition algorithm in detail in Section 4.3.

Feedback Delivery

Feedback information is a necessary component of any human-computer interaction. The functionality of feedback ranges from acknowledging a user's action to promote understanding of the system. It is well-established that humans can learn and improve performance of various motor control tasks using feedback information[Bil66].

Existing handwriting recognition systems do not provide sufficient feedback information for the users to understand and improve future performance. Normally, only two types of feedback are provided to the users. First, as the user writes on the touch screen, the system draws an ink trail at the locations of the touches to acknowledge that the input handwriting has been received. The second feedback is the recognition result which is usually given to user *after* the recognition process has ended. According to [FHM95], given only these types of feedback, the users did not know how to avoid future mistakes when the recognition had failed because they did not *understand* how the recognition system worked.

In the *co-adaptive* system, we provide the users with additional feedback that aids the user in the writing process. The additional feedback can be described based on their delivery timing as follows.

- **Immediate feedback** The immediate feedback is referred to the information given to the user continuously *during* the writing process [BM08]. The main purpose of this type of feedback is to continuously communicate to the user of what has been understood by the system so far and to allow the user to detect mistakes early. In this work, the immediate feedback is derived from the likelihood of each letter which is re-evaluated continuously as the user writes. Inferring from the likelihood information, the user can detect a mistake earlier (before the writing ends), or to stop writing when the likelihood indicates the correct intent. The delivery of this feedback can be done visually or acoustically.
- **Instructional feedback** The instructional feedback is given from time to time. The objective of this feedback is to automatically teach or guide the user to achieve a better recognition accuracy. The instructional feedback is derived from the personalized handwriting model using offline analysis and is given to the users in

various formats. For example, the system can point out two characters that are often confused with other and encourage the users to emphasize more on the difference between them.

4.2 System Architecture and Implementation

uRight is a personalized handwriting recognition system that we developed based on the idea of co-adaptation where both users and the interface itself can adapt to each other simultaneously. In the current state, uRight is presented in a format of “racing” game where, in each round, the user is presented with 10-15 randomly selected symbols, one symbol at a time. The objective of the game is that the user must write the symbol shown on screen as fast as possible. At the each of each round, the user will receive feedback in terms of a score indicating how many bits of information were successfully transferred per second (BPS) and a list of mistakes made by the recognizer. Figure 4.2 shows screenshots from the uRight game. It is important to note that the user can design their own gestures for any given symbol. This effectively allows the user to invent their own shorthand for different symbols.

The advantage of implementing uRight as a game is twofold. First, it allows us to control what the user is writing since the system specifically tells them what they need to write. Using this label information, we can easily collect labeled data for re-training of the handwriting model. Second, it provides an incentive for the users to regularly engage the system as they compete against their friends in the racing game. Consequently, this allows us to collect even more data for training the recognition system.

The uRight system consists of a centralized server that runs computationally expensive machine learning algorithms and Apple’s iOS client applications that implement the handwriting recognition capability. The central server implements the data collection and adaptation capabilities of the system. On the server, there is a separate account for each user. This decouples the users from the devices, allowing different users sharing the same device and a single user using multiple devices. The main functionality of the server is to collect and store handwriting examples that were sent by client applications. In our setup, the users are told which letter to write, so each example corresponds to a

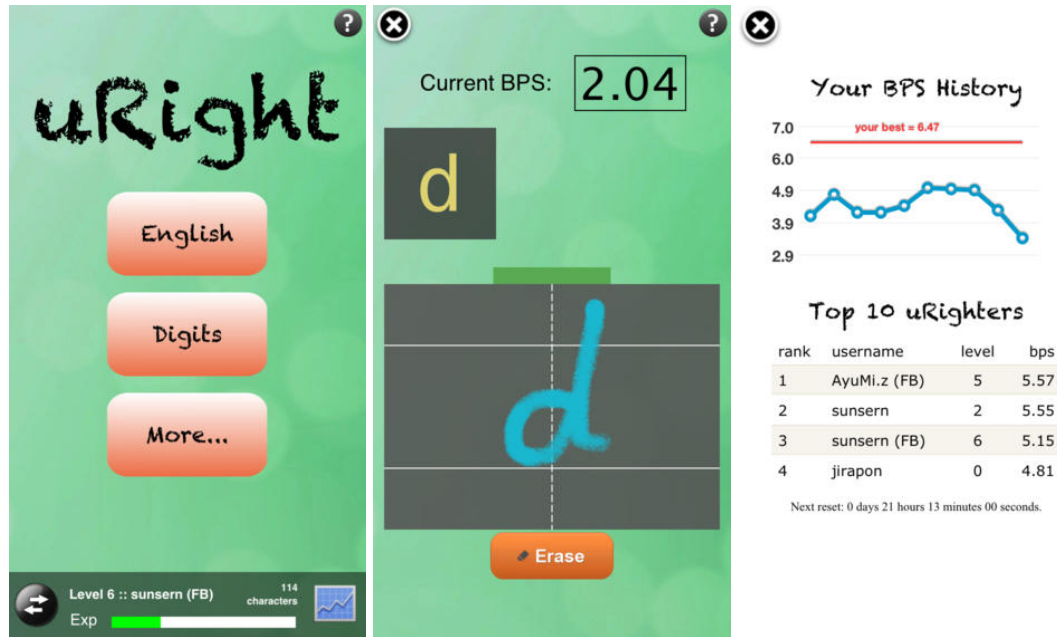


Figure 4.2: Screenshots from the uRight game. From left to right, the main menu, during each round, and the summary screen at the end of the round.

known letter. The server uses these labeled examples to update the handwriting model specifically for each user. The updated model is then pushed back to the device the next time the user uses the device.

The recognition interface is implemented as an iOS application that runs natively on Apple iPads and iPhones. The application is responsible for recognizing the user's handwriting in real time and for episodic communication with the central server. The device sporadically receives a set of character prototypes, personalized to each user from the central server. Based on these prototypes, the recognition algorithm finds the closest match to the current letter written by the user. This matching process is performed in real-time as the user is writing, allowing for more immediate feedback. Contrast this with other recognition systems such as Siri and Google goggles, that send each utterance/image to the a server which performs the recognition and sends back the result. Our system does not require a network connection for recognition and is able to respond within the time that the user is writing each character. Much of the development effort went into insuring a very fast response time while writing. Communication with the central server is needed only in order to update the prototypes and achieve better per-

sonalization. The iOS application is available for free from Apple's App Store ¹ as of April 2014 and the source code of the uRight system is also publicly available ².

Another advantage of basing our design on the client-server architecture is that prototypes can be shared between users. New users can therefore benefit from the information collected from long-term users that have a similar handwriting. This allows us to design a system that requires much less initial training. Figure 4.3 summarized the connectivity of various components of the uRight system

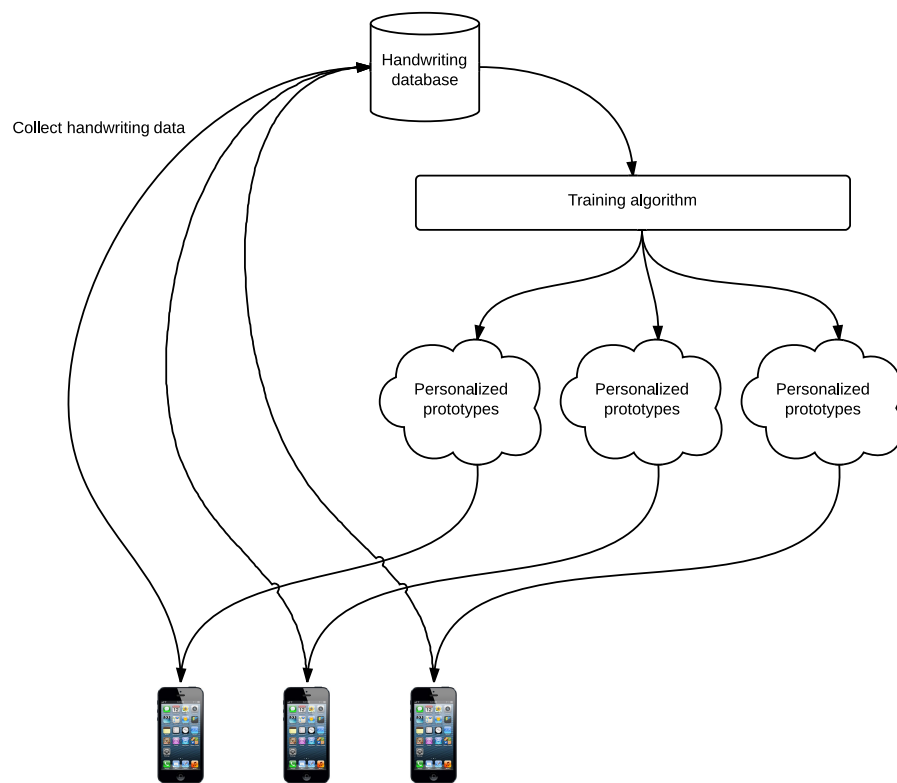


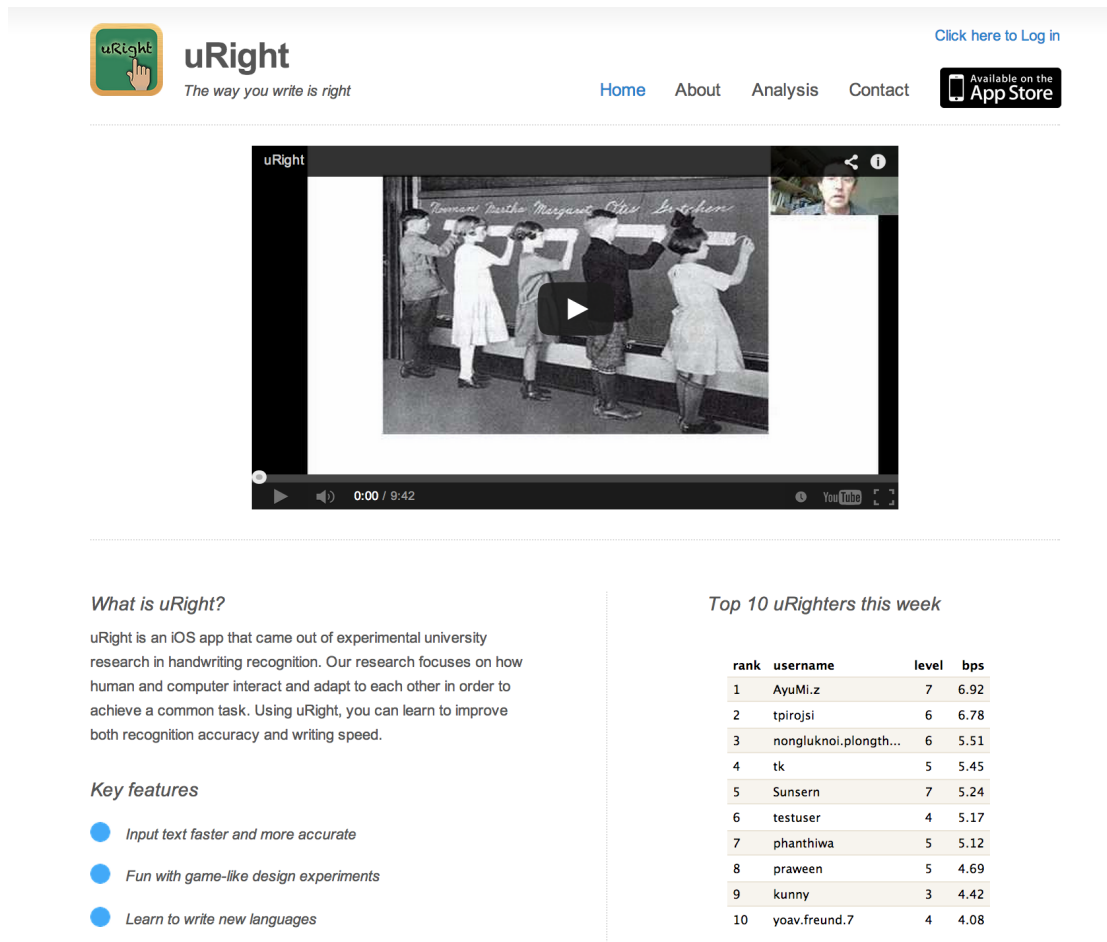
Figure 4.3: Connectivity of various components of the uRight system

We also developed a web interface and a scoreboard system to promote further user engagement. We found that the scoreboard system is effective at motivating the user

¹Download link from Apple App Store: <https://itunes.apple.com/us/app/uright/id642218957>

²The source code of uRight system is available from <https://github.com/sunsern/uright-ios-app>

engagement and enjoyable for the users. Through the web interface (Figure 4.4), the users can track their progress (Figure 4.5), view their prototypes over time (Figure 4.6), review recent mistakes and a confusion matrix (Figure 4.7- 4.8).



uRight
The way you write is right

Click here to Log in

Home About Analysis Contact

Available on the App Store

uRight

0:00 / 9:42

What is uRight?

uRight is an iOS app that came out of experimental university research in handwriting recognition. Our research focuses on how human and computer interact and adapt to each other in order to achieve a common task. Using uRight, you can learn to improve both recognition accuracy and writing speed.

Key features

- Input text faster and more accurate
- Fun with game-like design experiments
- Learn to write new languages

Top 10 uRighters this week

rank	username	level	bps
1	AyuMi.z	7	6.92
2	tpirojsi	6	6.78
3	nongluknoi.plongth...	6	5.51
4	tk	5	5.45
5	Sunsern	7	5.24
6	testuser	4	5.17
7	phanthiwa	5	5.12
8	praween	5	4.69
9	kunny	3	4.42
10	yoav.freund.7	4	4.08

Figure 4.4: Web interface of uRight

4.3 Adaptive Recognition Algorithms

This section outlines the recognition algorithm in the uRight system. We consider a particular variant of the handwriting recognition problem known as *online handwriting recognition*. The problem can be formalized as follows. Each input character X_i is defined by a sequence of strokes and each stroke is defined by a sequence of points

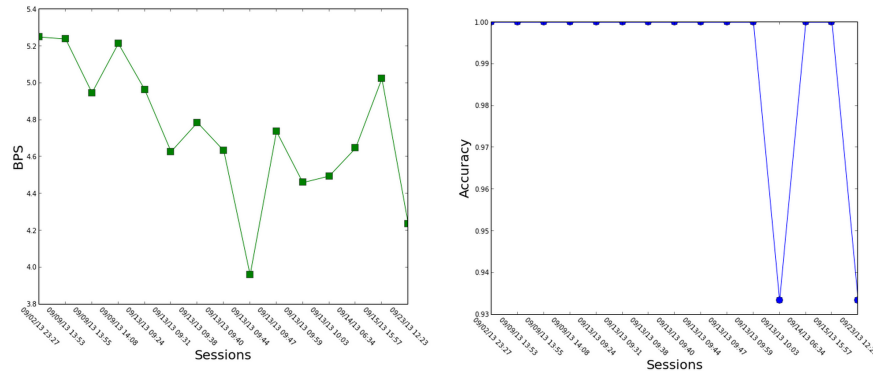


Figure 4.5: Example of bit-per-second (BPS) and accuracy plot over time of a user

(p_1, p_2, \dots, p_n) where p_i is a point on the xy -plane. Let Σ be the set of all possible labels. Given a training set $\mathcal{D} = \{(X_i, y_i \in \Sigma) : \forall 1 \leq i \leq m\}$, the task is to learn a classification function \mathcal{C} that maps an input instance X to its label y .

A number of machine learning algorithms have been applied to handwriting recognition. In general, they can be categorized into two types: *generative* and *discriminative*. The generative approach is based on constructing generative models, and the classification is often done by Bayesian inference. Hidden Markov models are commonly used to model handwritten characters [HBT96, CJ02, TG11]. This approach relies heavily on certain assumptions of the underlying model. The performance suffers when the assumptions fail to hold.

Discriminative approach, on the other hand, does not make such assumption and learns the classifier directly from the labeled training data. The discriminative learning algorithms have been applied to the this problem by many people [MGDV93, KC06, KAB08]. However, there is a fundamental issue with applying these algorithms directly to the problem of handwriting recognition. Most standard discriminative learning algorithms such as Neural Network and Support Vector Machine (SVM) [CV95, CST00], requires fixed-length input vectors but the handwritten data can have a variable length due to different writing speeds and sizes. An additional step must be taken in order to obtain a fixed-length feature representation. A naive solution would be to linearly scale the data so that they all have the same length, but the proper scaling factor is usually non-linear and not trivial. A better solution is to use a technique called *dynamic time*

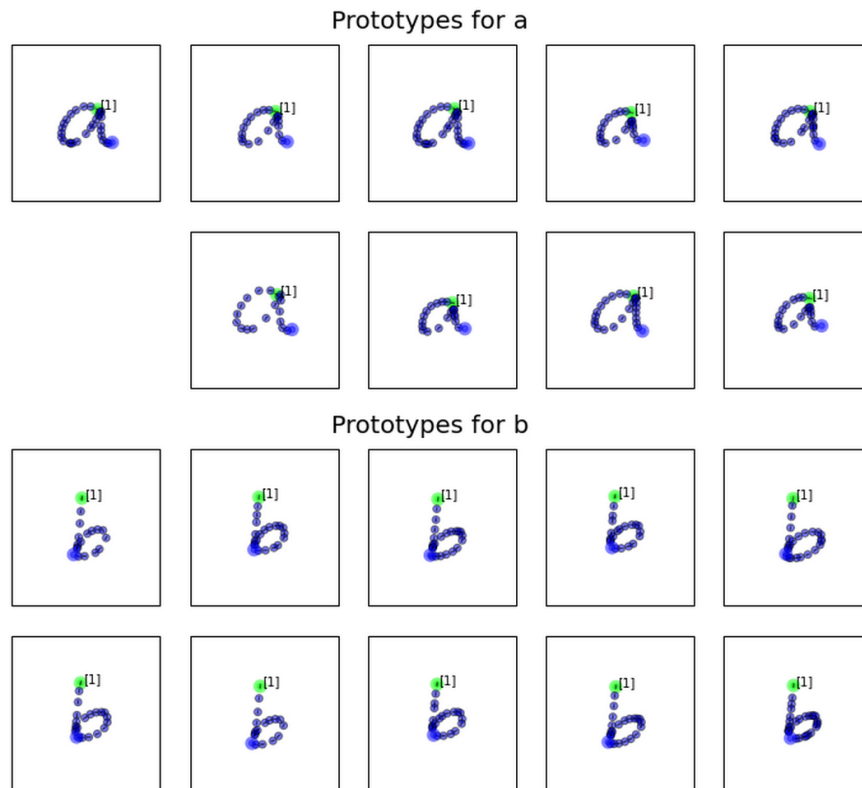


Figure 4.6: Prototypes over time for a user

warping [BHB02]. Another way is to develop an ensemble of global feature detectors, e.g. cusp detector or loop detector, and use their output as feature presentation for discriminative learning [JMW00]. Alternatively, there is *Fisher kernel* [JH99, TAKM04] that utilizes a generative model in order to generate fixed-length feature representation.

During the development of the uRight system, we have implemented two adaptive recognition algorithms from each of the categories. In the following sections, we discuss a discriminative algorithm that is based on SVM with Fisher kernel and then a generative algorithm that is based on a variant of nearest neighbor algorithm. It is worth noting that we ended up choosing the nearest neighbor algorithm over the SVM with Fisher kernel algorithm to include in the uRight system due to hardware limitation and the realtime recognition requirement.

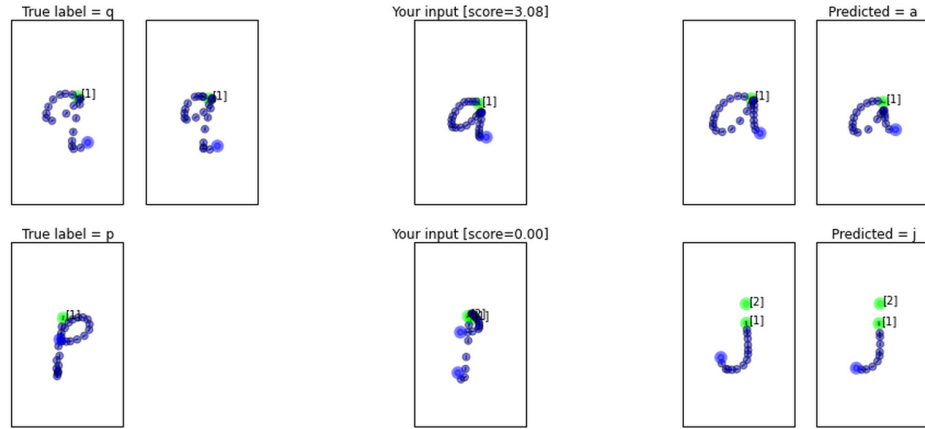


Figure 4.7: Mistakes made by a user

SVM with Fisher Kernel

Support vector machines (SVM) [CV95, CST00] are inherently developed for binary classification. However, there are various multiclass extensions to the SVM. Among them, 1-v-1 SVM is the most appealing to us because it allows us to perform pairwise analysis directly and also known to work well in practice [HL02]. The multiclass classifier generated from 1-v-1 SVM is basically a collection of binary classifiers between all possible pairs of classes. The final prediction is obtained by taking a majority vote.

Unlike traditional kernel functions, Fisher kernel [JH99, TAKM04] can operate on variable-length data. The idea behind Fisher kernel is that two similar sequences induce similar gradients in the space of the model's parameter. Formally, suppose $\mathcal{L}(\Theta|X)$ is the log-likelihood function of a generative model parameterized by Θ for an example X . Consider two input sequences X_i and X_j . Fisher kernel suggests that if X_i and X_j are similar then $\frac{\partial \mathcal{L}(\Theta|X_i)}{\partial \theta}$ and $\frac{\partial \mathcal{L}(\Theta|X_j)}{\partial \theta}$ will be also similar for every $\theta \in \Theta$. Let $\vec{g}_i = [\frac{\partial \mathcal{L}(\Theta|X_i)}{\partial \theta} : \forall \theta \in \Theta]$. Fisher kernel function is defined by

$$K(X_i, X_j) = \vec{g}_i^\top \mathbf{I}^{-1} \vec{g}_j \quad (4.1)$$

where $\mathbf{I} = \mathbf{E}_{\mathcal{D}} \left[\left(\frac{\partial \mathcal{L}(\Theta|X)}{\partial \theta} \right)^\top \left(\frac{\partial \mathcal{L}(\Theta|X)}{\partial \theta} \right) \right]$. The Fisher information matrix \mathbf{I} functions as a normalization factor for each parameter. The training process can be summarized below.

lem [HBT96, CJ02] and known to performs well. Each state of the HMM is represented by a multivariate Gaussian distribution with the following variables:

- Pen direction³
- Curvature.³
- Y-position.

Additionally, there is a special pen-up state where the emission probability is close to 1 when the input point is a pen-up and close to 0 otherwise. For state transitions, we allow transitioning from state i to state j where $j > i$. The number of states is fixed to 20 by cross-validation.

Suppose $\Pi, \mathcal{A}, \mathcal{B}$ be the state prior probabilities, the state transition probabilities and the state emission probabilities respectively. To estimate Π, \mathcal{A} and \mathcal{B} , we use Expectation-Maximization algorithm [Bil97] to maximize the log-likelihood of the training data \mathcal{D} . Note that the estimated parameters might not be optimal for the discrimination task [vdM11]. However, the advantage of maximizing the log-likelihood is that it can be done even without labeled data.

Computing Fisher Gradients

Once we have estimated the parameters $\Theta = (\Pi, \mathcal{A}, \mathcal{B})$ of the HMM, we can compute the log-likelihood gradient vector \vec{g}_i for each example X_i . For our system, the gradient vector \vec{g} only contains partial derivatives with respect to $p_{ik} \in \Pi$ and $a_{pq} \in \mathcal{A}$.

Given a training character $X = (p_1, p_2, \dots, p_T)$. Suppose N is the number of states in the HMM. Let $\alpha_i(t) = P(X, S_t = i | \Theta) \forall 1 \leq i \leq N$ be the probability of being in state i at time t and $\mathcal{B}_j(p)$ is the probability of state j generating observation p . The gradient vector $\vec{g}(X)$ with respect to $\pi_k \in \Pi$ and $a_{pq} \in \mathcal{A}$ can be written as

$$\vec{g}(X) = \left[\sum_{j=1}^N \frac{\partial \alpha_j(T)}{\partial \theta} : \forall \theta \in \Pi \cup \mathcal{A} \right] \quad (4.2)$$

³For calculation details, please refer to Jaeger et al. [JMW00]

where

$$\frac{\partial \alpha_j(t+1)}{\partial \pi_k} = \left[\sum_{i=1}^n \frac{\partial \alpha_i(t)}{\partial \pi_k} a_{ij} \right] \mathcal{B}_j(p_{t+1})$$

$$\frac{\partial \alpha_j(1)}{\partial \pi_k} = \begin{cases} \mathcal{B}_j(p_1) & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

and

$$\frac{\partial \alpha_j(t+1)}{\partial a_{pq}} = \begin{cases} \left[\sum_{i=1}^n \frac{\partial \alpha_i(t)}{\partial \pi_k} a_{ij} \right] \mathcal{B}_j(p_{t+1}) & \text{if } q = j \\ \quad + \alpha_p(t) \mathcal{B}_j(p_{t+1}) & \\ \left[\sum_{i=1}^n \frac{\partial \alpha_i(t)}{\partial \pi_k} a_{ij} \right] \mathcal{B}_j(p_{t+1}) & \text{otherwise} \end{cases}$$

$$\frac{\partial \alpha_j(1)}{\partial a_{pq}} = 0, \forall 1 \leq j \leq N$$

After we have computed \vec{g}_i for every training example, the Fisher kernel can be computed by using Equation 4.1.

Training Personalized Classifier

To personalize the SVM classifier, we use different example weights for the user data and for the other users' data. Suppose we want to train a classifier for user k . Let ρ be the ratio of the weight of each example from user k to the weight of each example from other users. As ρ increases, the classifier becomes more and more specific to the data from user k . When $\rho = 0.0$, the user's data has zero weights, this is equivalent to *writer-independent* classifier. On the other hand, when $\rho = \infty$ the other users' data gets zero weights, this is equivalent to a purely-personalized classifier.

To do so, we modify the objective function for SVM slightly.

$$\min \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \omega_i \xi_i \quad (4.3)$$

where ω_i is the weight for example i . For any given ρ , the classifier for user k , C_k^ρ can be obtained by optimizing Equation 4.3 where

$$\omega_i = \begin{cases} 1.0 & \text{if example } i \text{ does not belong to user } k \\ \rho & \text{if example } i \text{ belongs to user } k \end{cases}$$

Note that, if $\rho = \infty$, then let $\omega_i = 0$ when example i do not belong to use k and $\omega_i = 1$ otherwise.

For the SVM implementation, we use the weighted version of C-SVM in the LIB-SVM library, which is a well-known public implementation of SVM [CL11]. To train a 1-v-1 classifier for each of the 26 lowercase English characters, we need to train a total of $\binom{26}{2} = 325$ binary classifiers. For each of the classifier, prior to the actual training, we perform a 5-fold cross-validation to find the best regularization parameter C via a simple grid-search method in the range of $[0.1, 5.0]$. Then, each binary classifier is trained with the best value of C .

In Figure 4.9 we show the performance of classifiers generated using different values of ρ . the data is split into training and test set randomly and both training and test data are weighted using the same value of ρ .

The results for $\rho = \infty$ (data only from the individual user) and $\rho = 0$ (data only from other users) are as expected. Using data from other users is initially better, because the training set is larger. Somewhere around 9 examples/character there is a transition where using only the personal data becomes better. The graphs for $\rho = 1, 2, 4, 8$ are largely overlapping implying that setting ρ to any of these non-extreme values works reasonably well.

DTW-based Nearest Neighbor Algorithm

We implemented another recognition algorithm based on the nearest neighbor algorithm with dynamic time warping divergence. At a high-level, the recognition algorithm can be outlined as follows. For each user, we create and maintain one or more character models for each character in \mathcal{E} . We refer to each of such models as a *prototype*. Each prototype is basically a representative handwriting instance from the user. Technically, the prototypes can be viewed as left-to-right hidden Markov models with Gaussian observation [TG11]. Let \mathcal{P}_u denote the set of prototypes for a user u . The adaptivity of our system comes directly from the fact that \mathcal{P}_u is modified over time. In the decoding process, given a handwriting trajectory and a set of prototypes \mathcal{P}_u , the system computes a posterior distribution Y_{final} and, when a single prediction is needed, the element with the maximum likelihood is predicted.

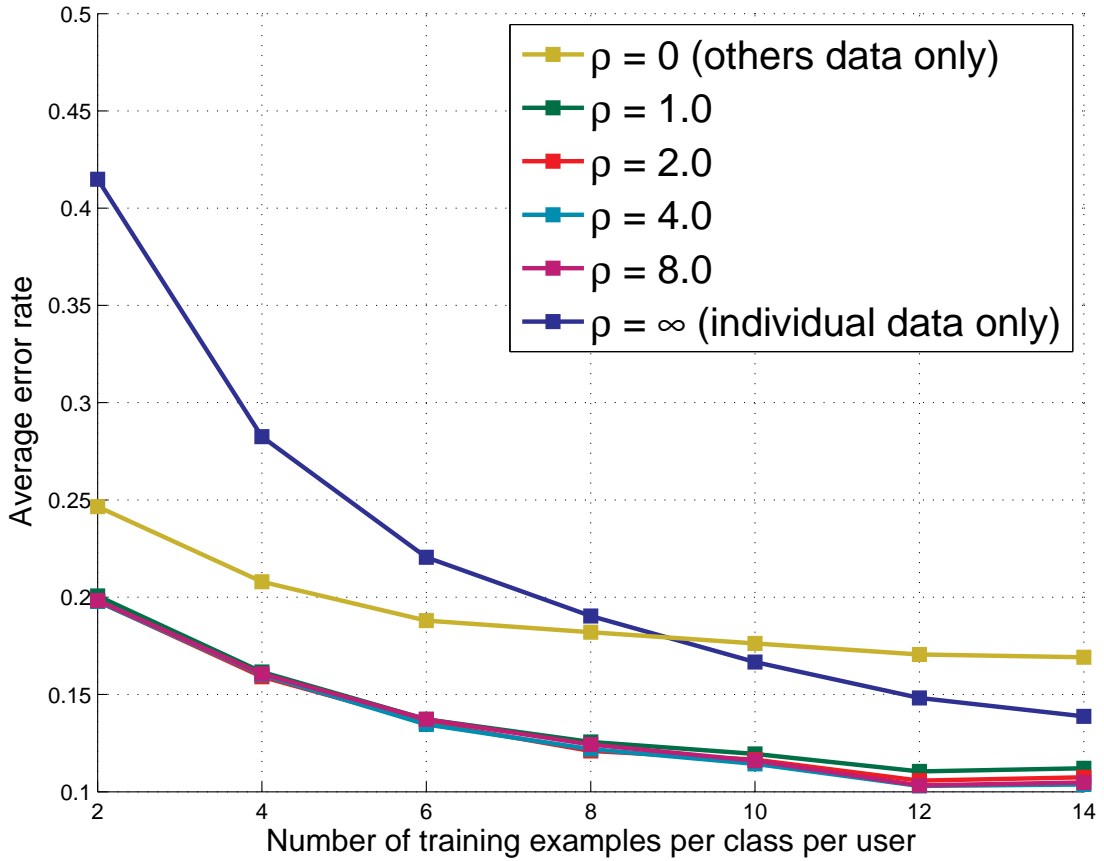


Figure 4.9: Average error rate as a function of the training set size and the value of the mixing parameter ρ . The horizontal axis corresponds to the number of training examples per character for the individual user. The order of the data is randomized to remove the effect of user adaptation.

Feature vectors and distance function

In addition to the x- and y-coordinate, each handwriting trajectory is supplemented with writing direction information. Specifically, each handwriting instance is represented by a sequence of feature vectors $\langle f_1, \dots, f_T \rangle$ where $f_i = (x_i, y_i, dx_i, dy_i)$. (x_i, y_i) denotes the normalized touch-screen coordinate and

$$(dx_i, dy_i) = \left(\frac{x_i - x_{i-1}}{z}, \frac{y_i - y_{i-1}}{z} \right), z = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

denotes the writing direction.

To measure the similarity between two handwriting instances, we use *dynamic*

time warping (DTW) divergence [RJ93] as the distance function in our algorithm. The DTW measure is commonly used for variable-length data such as handwriting and speech. The calculation can be done efficiently using dynamic programming.

Initial adaptation

The initial adaptation is critical for any intelligent system. It is unquestionable that the performance of any well-behaved intelligent system increases as the system learns more about the user. If the initial adaptation is poor, the users might get frustrated with the system and stop using it even before it can fully adapt to them.

We address the problem of initial adaptation by sharing data across different users. Typically, people do have similar handwriting especially when they share the same educational culture. The process of the initial adaptation can be described as follows. In the very first interaction with the user u , our system has no information about the user and, therefore, assign a set of typical prototypes which has been trained using data from multiple users in the past. Specifically, the typical prototypes are the centroids of the clusters returned by running a clustering algorithm (k-means) on a set of training handwriting instances. We refer to this set of prototypes as \mathcal{P}_0 . After the first interaction, the system creates a new set of prototypes $\mathcal{P}_{(u,1)}$ by recomputing the centroids of the clusters after adding the examples from the user to the pool with significantly higher weights than the rest. For our dataset, we found that, at most 2 prototypes are sufficient for each user-character pair.

Adapting the prototypes over time

After collecting a few examples of the user's handwriting, the system again performs the weighted clustering algorithm on the data to generate a new set of prototypes $\mathcal{P}_{(u,i+1)}$. In this stage, only examples from the user and previous prototypes are considered. This adaptation process happens after 3-5 new examples are acquired.

To improve real-time performance, we need to keep the lengths (number of states) of the prototypes as small as possible. After the new prototypes are chosen, the system performs an additional step to shorten the length of each prototype. This pruning process is similar in spirit to removing and merging unnecessary hidden states

in an HMM. The basic idea is to remove unwanted states while maintaining the same recognition power using a variant of forward-backward algorithm [Bil97]. Figure 4.10 shows the hidden states before and after the reduction step.

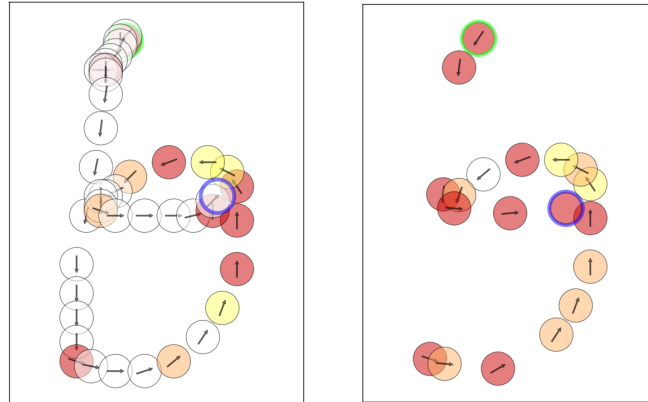


Figure 4.10: The hidden state reduction process is applied to each prototype to remove rarely visited states with respect to the training set. The originally trained prototype is shown on the left and the reduced prototype is shown on the right. The intensity of the colors corresponds to the expected number of times the state being mapped to.

Decoding

Our decoding algorithm is based on the standard Bayesian inference. Namely, given a trajectory $W_{1:T}$ and the current set of prototypes \mathcal{P}_u , the algorithm computes the distance from $W_{1:t}$ to each of the prototypes in \mathcal{P}_u for all $1 \leq t \leq T$. The distances are then transformed into a probability distribution Y_t . We use e^{-x} as the transfer function. When a single prediction is expected, the algorithm simply returns the prediction with the maximum likelihood.

Algorithm 4 Beam-search forward algorithm

Initial Assumptions: $\{T_1, \dots, T_m\}$ = the prototype set

K = beam width

$A_{i,j,k}$ = the transition probability from state j to state k of prototype T_i

$B_{i,j}(f_t)$ = the emission probability of state j of prototype T_i with observation f_t

- 1: $\alpha_{i,1}^{(0)} \leftarrow$ the prior probability of prototype T_i
 - 2: $active \leftarrow \{\alpha_{i,0}^{(0)}; \forall 1 \leq i \leq m\}$
 - 3: **for** each time step t **do**
 - 4: $new_active \leftarrow \{\}$
 - 5: **while** $active \neq \emptyset$ **do**
 - 6: remove $\alpha_{i,j}^{(t-1)}$ from $active$
 - 7: **for** $0 \leq k \leq 2$ **do**
 - 8: $\alpha_{i,j+k}^{(t)} \leftarrow \alpha_{i,j}^{(t-1)} A_{i,j,j+k} B_{i,j+k}(f_t)$
 - 9: insert $\alpha_{i,j+k}^{(t)}$ to new_active or add to the existing value
 - 10: **end for**
 - 11: **end while**
 - 12: $active \leftarrow top_states(new_active, K)$
 - 13: **end for**
-

Chapter 5

Co-adaptation in Handwriting Recognition

Handwriting is a natural and versatile method for human-computer interaction, especially on small mobile devices such as smart phones. As handwriting varies significantly from person to person, it is difficult to design a handwriting recognition system that performs well for all users. Modern handwriting recognizers resort to machine learning methods to adapt and specialize their handwriting models to each individual user.

Most machine learning methods are developed under the assumption that the data is generated by an oblivious process that does not change in reaction to the learning process. However, many applications of machine learning in Human Computer Interaction (HCI), such as speech recognition, handwriting recognition and gesture recognition do not obey this rule. In these cases the user is participating in an interaction where they can provide corrective feedback when their intent is misunderstood. As a result, the user is motivated to subsequently change their speech, handwriting or gestures so as to avoid errors and speed up the interaction. We call this learning situation *co-adaptation* to emphasize that the human and the computer are adapting to each other in parallel.

In general, co-adaptation can manifest in any adaptive system. Designing a system that co-adapts with the users is a challenging problem on its own [H00, Mae94, LD09]. Our goal in this chapter is not to address those challenges, but rather to focus on characterizing the impact of machine adaptation and of human adaptation in the con-

text of handwriting recognition. We believe that this study will provide us with useful insights towards designing a more efficient adaptive handwriting recognition system.

In order to evaluate performance of a handwriting recognition system under co-adaptation, we introduce a framework based on the idea of Shannon’s communication channel [Sha48] that considers both the user and the handwriting recognizer in a single system. Under this framework, we define the notion of “channel rate” that measures the amount of information successfully transferred from the user to the computer.

To quantify the effect of machine adaptation and of user adaptation empirically, we developed a handwriting recognition system that is capable of adapting to the handwriting of each individual user over time. We collected usage data from 15 different users and performed an analysis of the channel rate.

This chapter is organized as follows. First, in Section 5.1, we present the information-theoretic framework for quantifying the efficiency of a handwriting system where the system includes both the user and the computer. In Section 5.2, we describe the experiment and present the results in terms of the performance measures derived from the proposed framework. Finally, we draw some conclusions in Section 5.4.

5.1 Handwriting Recognition as a Communication Channel

Unlike typing, which transmits information to the computer at discrete time points, handwriting continuously transmits information as the writer creates the trajectory. Traditionally, handwriting data is analyzed one “unit” at a time where “unit” can be a stroke, a character, a word or even a sentence. In this work, we propose an alternative analysis where the data is analyzed in fixed intervals of time. We consider the process of writing as a process through which the intended letter is disambiguated from the other possible letters.

We formalize this process using the concept of communication channel [Sha48]. Let \mathcal{E} denote the set of all possible input. Technically, the set \mathcal{E} can be a set of sentences, a set of words, or a set of characters. Without loss of generality, in this work, we assume that \mathcal{E} is a set of 26 English characters. We also ignore dependencies between characters

due to the language model and due to the co-articulation effects between neighboring handwritten characters.

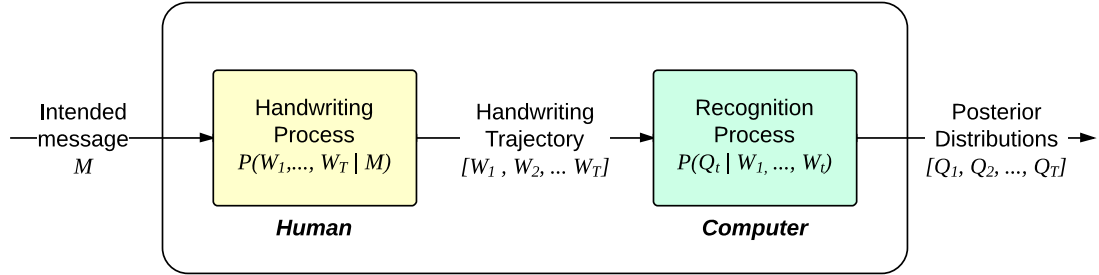


Figure 5.1: Handwriting recognition channel.

As shown in Figure 5.1, the channel is comprised of two separate processes. First, the handwriting process is the process of which the user translates an intent $M \in \mathcal{E}$ into a series of hand movements which is sampled at some rate to create a discrete time trajectory: $W_{1:T} = [(x_1, y_1), \dots, (x_T, y_T)]$. In other words, this process *encodes* the intent M into a trajectory $W_{1:T}$. Let \bar{W} denote the entire trajectory vector. The distribution $P(\bar{W}|M)$ denotes the variability of the encoding process. The second process is the recognition process that decodes the handwriting trajectory back into the original intent. For each time step t where $1 \leq t \leq T$, the process maps a trajectory $W_{1:t}$ to a distribution over \mathcal{E} , denoted by \mathcal{Q}_t .

Let T_{final} and $\mathcal{Q}_{\text{final}}$ denote the final writing duration and the posterior distribution when the user finishes writing the trajectory. According to the theory of channel capacity, the information transmitted through the channel can be quantified by the mutual information between the input M and the decoding posterior $\mathcal{Q}_{\text{final}}$, denoted by $I(M; \mathcal{Q}_{\text{final}})$. We define the mean posterior of $\mathcal{Q}_{\text{final}}$ conditioned on M and the average posterior distribution as follows.

$$P(\mathcal{Q}_{\text{final}}|M) = \int_{\bar{W} \sim P(\bar{W}|M)} P(\mathcal{Q}_{\text{final}}|\bar{W})P(\bar{W}|M)$$

$$P(\mathcal{Q}_{\text{final}}) = \sum_{m \in \mathcal{E}} P(M = m)P(\mathcal{Q}_{\text{final}}|M = m)$$

Given these two expressions, we define the mutual information between the character M and the decoding $\mathcal{Q}_{\text{final}}$ as

$$I(M; \mathcal{Q}_{\text{final}}) = H(\mathcal{Q}_{\text{final}}) - \sum_{m \in \mathcal{E}} P(M = m)H(\mathcal{Q}_{\text{final}}|M = m)$$

where the entropy of $\mathcal{Q}_{\text{final}}$ is defined as

$$H(\mathcal{Q}_{\text{final}}) = - \sum_{m \in \mathcal{E}} P(\mathcal{Q}_{\text{final}} = m) \log_2 P(\mathcal{Q}_{\text{final}} = m)$$

Next, we can define the channel rate in terms of the mutual information and expected writing duration as

$$R_{\text{MI}} = \frac{I(M; \mathcal{Q}_{\text{final}})}{\mathbb{E}[T_{\text{final}}]} \quad (5.1)$$

However, the channel rate R_{MI} is not suitable for practical implementation for two reasons. First, the estimation of $H(\mathcal{Q}_{\text{final}}|M)$ requires an extensive amount of data. Secondly, suppose the original intent is m , R_{MI} yields a high value as long as $P(\mathcal{Q}_{\text{final}}|M = m)$ concentrates any single intent n even when $n \neq m$. Thus, we propose an alternative measure to the R_{MI} based on the idea of log loss, called R_{LL} . We define R_{LL} to be

$$R_{\text{LL}} = \frac{H(\mathcal{Q}_{\text{final}}) - \sum_{m \in \mathcal{E}} P(M = m)(-\log_2 P(\mathcal{Q}_{\text{final}} = m|M = m))}{\mathbb{E}[T_{\text{final}}]} \quad (5.2)$$

The relationship between R_{MI} and R_{LL} is worth noting. When $(-\log_2 P(\mathcal{Q}_{\text{final}} = m|M = m))$ is small then the conditional entropy $H(\mathcal{Q}_{\text{final}}|M)$ is also small. As a result, the mutual information $I(M; \mathcal{Q}_{\text{final}})$ will be close to its maximal possible value of $H(\mathcal{Q}_{\text{final}})$. In other words, the log loss term $(-\log_2 P(\mathcal{Q}_{\text{final}} = m|M = m))$ provides an upper bound for the conditional entropy $H(\mathcal{Q}_{\text{final}}|M)$ up to some constant factor. For the remaining of this chapter, when we refer to the *channel rate*, we strictly refer to R_{LL} .

Intuitively, the channel rate is a measure that quantifies both accuracy and speed of a handwriting recognition channel at the same time. Handwriting, as well as many other motor control tasks, obeys the speed-accuracy tradeoff [Fit54]. It is not sufficient to quantify the efficiency of a handwriting recognition system by its recognition accuracy alone. For example, a system that requires the user to write each character in a specialized form may attain a very high recognition accuracy, but it would require the

user more time and effort to use. Such system might not be as efficient as a system that makes more errors but allows the user to write freely. This leads us to believe that the channel rate is a suitable measure that any handwriting recognition system should aim to maximize. In a sense, maximizing the channel rate is equivalent to finding a balance between maximizing the recognition accuracy and minimizing the writing time and effort of the user.

Based on this framework, it follows that the channel rate can be improved by a combination of human learning and machine learning, which corresponds to improving the handwriting process and the recognition process respectively. Ideally, $\mathcal{Q}_{\text{final}}$ will always be concentrated on the original intent M . This would mean that the channel is perfect and works without error. However, in real-world scenarios, errors will occur. One source of errors comes from mistakes made in the recognition process. These recognition errors can be reduced using training data and machine learning. The harder problem is when there is a significant overlap between $P(\bar{W}|M)$ for different intents. In this situation, we will need to rely on the user to make their handwriting less ambiguous. Although the effect of human learning is always present, we believe that it can be enhanced by giving useful feedback to the user in the form of guidance or lessons.

5.2 Experiment

The main objective of our experiment is to determine and quantify the effect of machine adaptation and of human adaptation when the users interact with the system over some period of time. We implemented the handwriting recognition system described in Chapter 4. The system was presented to the users as a writing game. In each session, each participant was presented with a random permutation of the 26 lowercase English alphabets i.e. $\mathcal{E} = [a \dots z]$ and $P(M)$ is uniform. The objective of the game was to write the presented characters as quickly as possible and, more importantly, the handwritten characters should be recognizable by the system. A score, which is the average *channel rate* of the session, was given to the user right after each session to reflect the performance of the session. There were 15 participants in this experiment. We asked them to play our game for at least 20 sessions over multiple days in his/her

own pace. We did not control past experience of the participants. Some of them had more experience with touch screens than others.

The experiment was set up to demonstrate a condition called *co-adaptation* where both the user and the computer were allowed to adapt together. We denote this condition R_{adapt} . To investigate the effect of co-adaptation, we create a controlled condition called R_{fixed} where the computer was not allowed to adapt with the user. In other words, we ran a simulation to figure out what the channel rates would have been if the prototype sets were never changed from \mathcal{P}_0 . Ideally, it would be more preferable to have R_{fixed} determined by another control group where the prototypes were kept fixed and never changed. However, the results from the simulated condition can be seen as a lower bound on the amount of the improvement attributable to human learning and, therefore, it is sufficient to demonstrate our point.

5.3 Results and Discussion

The average channel rates per session of the two conditions R_{adapt} and R_{fixed} are shown in Figure 5.2a and Figure 5.2b respectively. In both conditions, the results show increases of the channel rate over time where the improvement in the early sessions seems to be larger than in the later sessions. Figure 5.2c shows the difference of R_{adapt} and R_{fixed} which corresponds to the channel rate of the system when we ignore the effect of user adaptation. From the result, we observe that the impact of machine adaptation tapers off after 10 sessions.

Although the prototype set was not changing in R_{fixed} , we observe that channel rate increases over the sessions. To quantify our confidence to this increase, we perform the paired t-test to compare the difference between the average channel rate in the first 5 sessions and in the last 5 sessions. We find that the difference is statistically significant with p -value < 0.0011 . This suggests that the users improve the handwriting on their own even without machine adaptation. In other words, the effect of *user adaptation* is indeed significant.

Furthermore, Figure 5.3a and Figure 5.3b reveal that the major contribution of *user adaptation* comes from the fact that the users write faster in the last 5 sessions

compared to the first 5 sessions ($p < 0.0001$), and not because of the system received more information from the user ($p = 0.9723$). This result is as expected according to the law of practice [NR81].

We also perform per-user analysis of the channel rate. In Figure 5.4a, we compare R_{adapt} and R_{fixed} for each user. We find that the channel rate of R_{adapt} is significantly higher than that of R_{fixed} with $p < 0.0006$. This result confirms that the machine adaptation helps improving the overall channel rate. In addition, we calculate the theoretical maximum of the channel rate under the assumption of the perfect recognition, denoted by R_{ideal} . The maximum rates are given by $H(\mathcal{Q}_{\text{final}})/\mathbb{E}[T_{\text{final}}]$ and we approximated $H(\mathcal{Q}_{\text{final}}) = \log_2(26)$.

In the case of perfect recognition, a simple way to increase the channel rate is to expand the character set \mathcal{E} to include more symbols. However, in reality, doing so can lead to a recognition error rate which impairs the channel rate. An interesting future direction is to design a character set that would maximize the channel rate. Figure 5.4b reveals the efficiency of each letter for our handwriting channel. Characters with complex stokes, such as 'q', 'g', 'k', are not as efficient as characters with simple strokes such as 'c', 'o', 'l'. While this finding is not surprising, it implies that, for a handwriting system to be truly efficient, it must allow the user to write in a less complex style while not losing recognition accuracy. How to exactly design such system is still an open problem and requires a more elaborate study.

Confusion and the conditional entropy

In addition to the experiment, we performed a detailed analysis on the recognition errors made by the system. Specifically, we computed a confusion matrix based on the data from the experiment. The confusion matrix indicated that 99% of the mistakes concentrate among 33 pairs of prototypes out of the total of 2278 pairs. This suggests that the confusions only happen between a few pairs of prototypes. Figure 5.5 shows some of the confusion pairs and the handwritten examples that were misrecognized. By inspection, we found that the confused handwritten characters were very similar for some letter pairs such as 'n'-'u', 'n'-'h' or 'r'-'v'.

The confusion is closely related to the conditional entropy $H(\mathcal{Q}_{\text{final}}|M)$. When

this is no confusion, the entropy quickly converges to zero as demonstrated in Figure 5.6. This suggests that early termination of the writing is viable. The system could have notified the user to stop writing at $\mathbb{2}$ and it can still recognize the partial handwriting as a 'z'. On the other hand, when there is a confusion, the entropy does not necessarily converge to zero when at the end of the writing e.g. the entropy of 'y' in Figure 5.8c.

In Figure 5.8, we look closely at the evolution of $P(\mathcal{Q}_t|W_{1:t})$ of a confusable triplet: 'g', 'y' and 'q'. In Figure 5.8a, the probability of 'g' starts to dominate other contenders e.g. 's' and 'a' after $\mathbb{3}$. Similarly, in Figure 5.8b, the posterior distribution evolves similarly to what we observe in Figure 5.8a then the probability of 'q' increases towards the end of the handwriting. This indicates that the crucial information that distinguishes between 'g' and 'q' is concentrated towards the end of the trajectory. Based on 5.5, the system sometimes confuses 'y' with 'g'. We suspect that such confusion happens when the probability of 'y' between $\mathbb{1}$ and $\mathbb{2}$ is too small relative to the probabilities of the contenders. The posterior distribution of a correctly recognized 'y' is shown in Figure 5.8c.

In Figure 5.7, we show the posterior distributions over time of 3 examples selected from a single user: a correctly recognized 'n', a correctly recognized 'h' and an 'n' that was recognized as an 'h'. We notice that, when the system correctly recognized an 'n', the probability of 'n' increases significantly between $\mathbb{2}$ and $\mathbb{4}$, which corresponds to the upward movement of the hand when writing both 'n' and 'h'. This information can be delivered to the user in a form of the instructional feedback to encourage the user to pay more attention to the upward movement part when writing the pair.

5.4 Conclusions

We presented an information-theoretic framework for quantifying the information rate of a system that combines a human writer with a handwriting recognition system. Using the notion of channel rate, we investigated the impact of machine adaptation and human adaptation in an adaptive handwriting recognition system. We analyzed data collected from a small deployment of our adaptive handwriting recognition system and concluded that both machine adaptation and human adaptation have significant impact on

the channel rate. This result led us to believe that, for a handwriting recognition system to achieve the maximum channel rate, both machine adaptation and human adaptation are required and must be present together. Specifically, such system should be able to adapt to the user and, at the same time, allow the users to write or scribble using simple hand movement as improving writing speed is crucial for attaining a higher channel rate. Additionally, the system should have a mechanism to giving feedback to the user when their handwriting cannot be recognized.

Acknowledgements

This chapter is based on unpublished work that is currently in submission as of the writing of this thesis. It is joint work with Yoav Freund. The dissertation author is the primary investigator and author of this work.

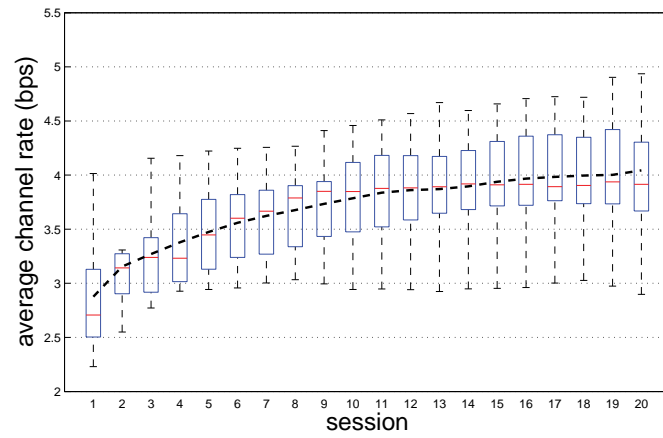
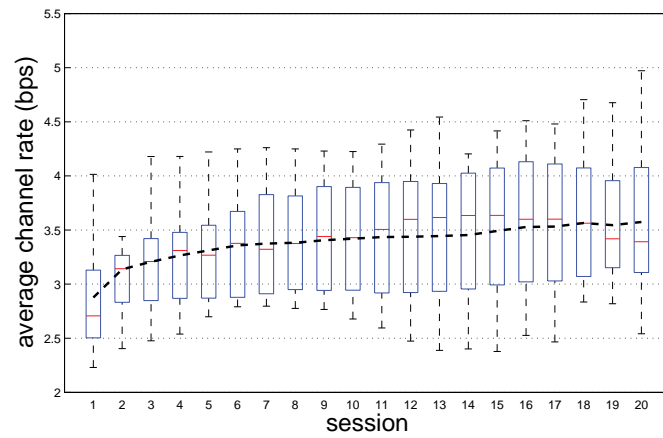
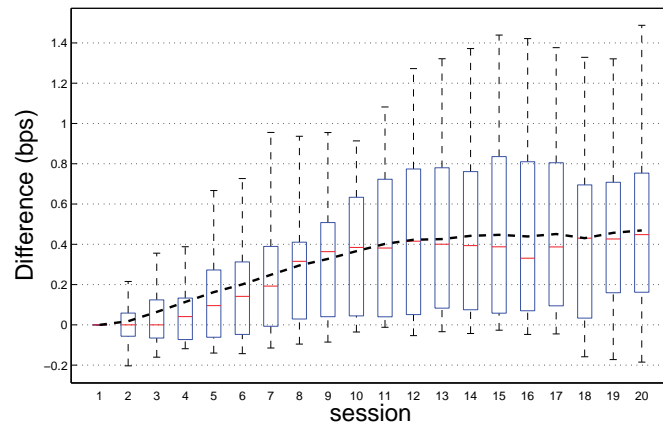
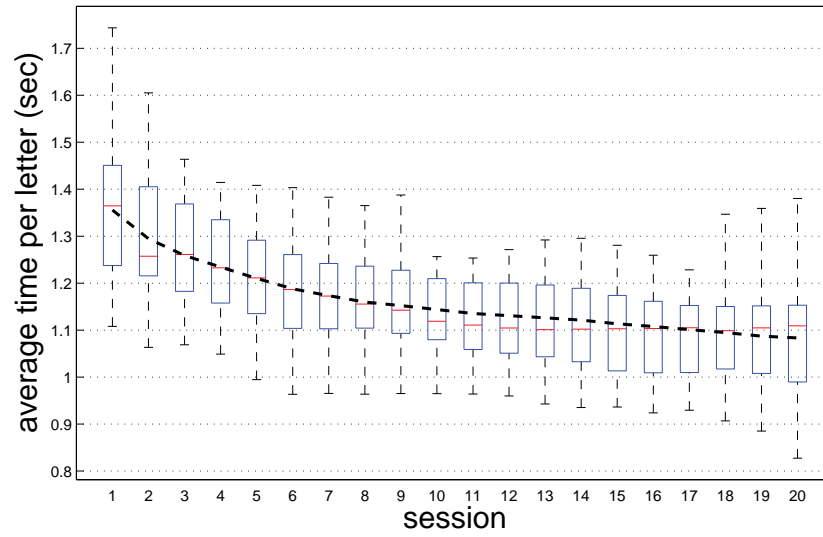
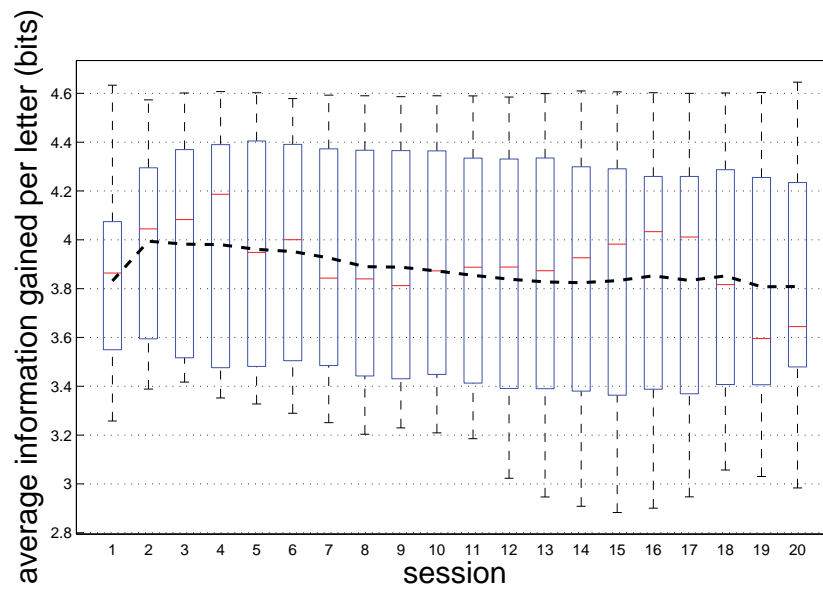
(a) R_{adapt} (b) R_{fixed} (c) $R_{\text{adapt}} - R_{\text{fixed}}$

Figure 5.2: Channel rate per session of all users with (5.2a) and without (5.2b) presence of machine learning.

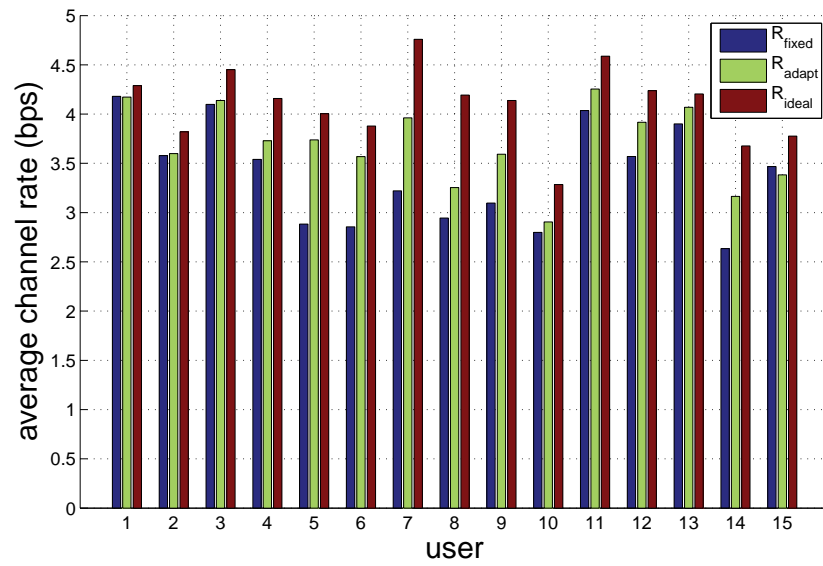


(a) Writing duration

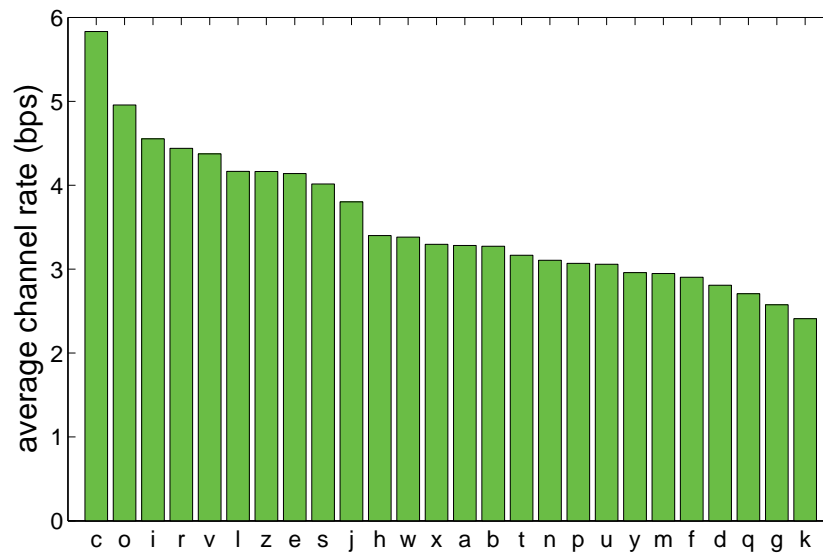


(b) Mutual information

Figure 5.3: Average writing time per session and the average mutual information per session under the condition R_{fixed} .



(a)



(b)

Figure 5.4: Average channel rates. (a) The average channel rate of each user in R_{adapt} and R_{fixed} . R_{ideal} shows the maximum channel rate possible given the average writing speed of each user. (b) Average channel rate of each character under the condition R_{adapt} .

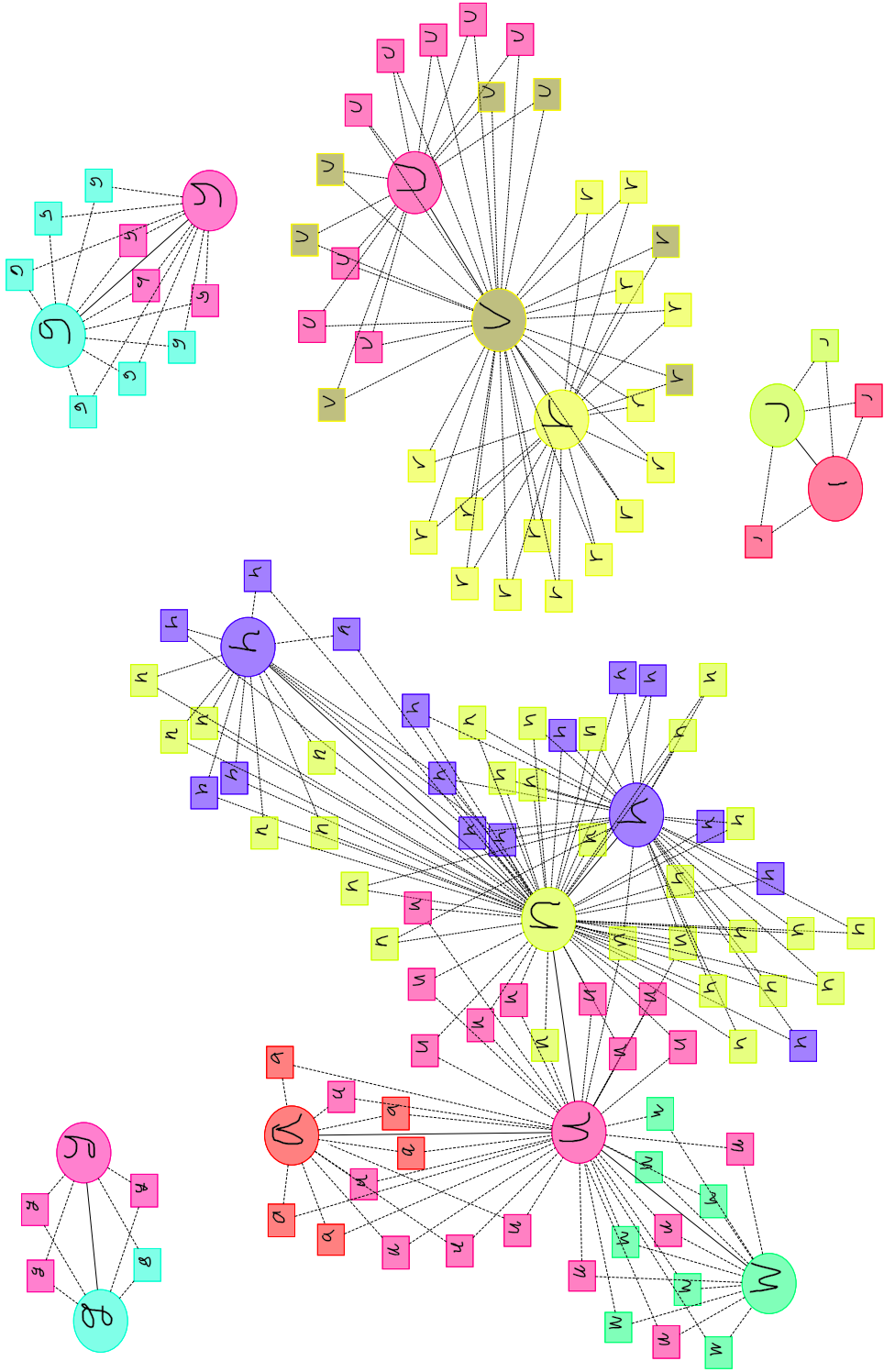


Figure 5.5: Confusion between two prototypes (circle nodes) is represented by an edge. Some of the confusing examples are shown in square nodes. Only confusable prototypes are shown.

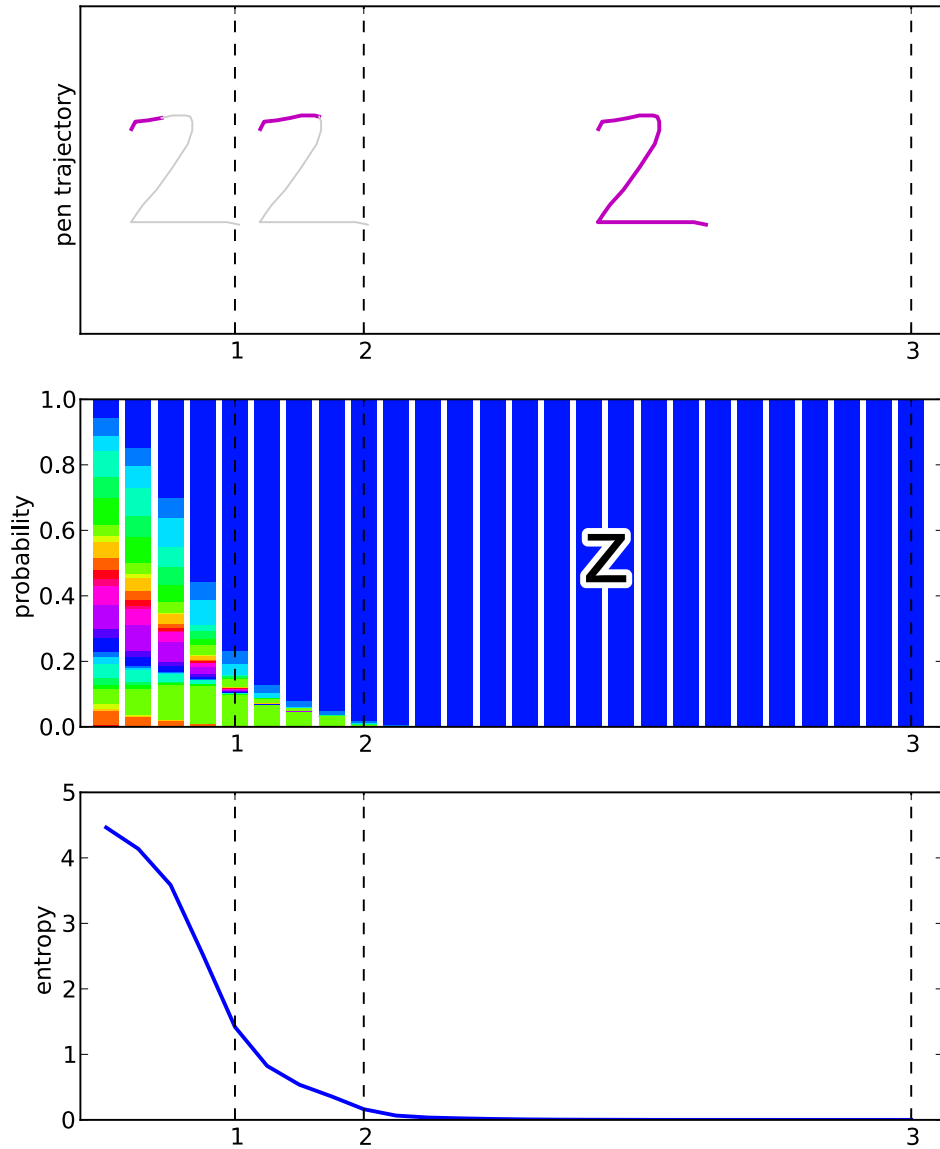


Figure 5.6: Conditional entropy of a “z” . The conditional entropy $H(\mathcal{Q}_{\text{final}}|M)$ quickly reduces to 0 when there is no confusion with other prototypes.

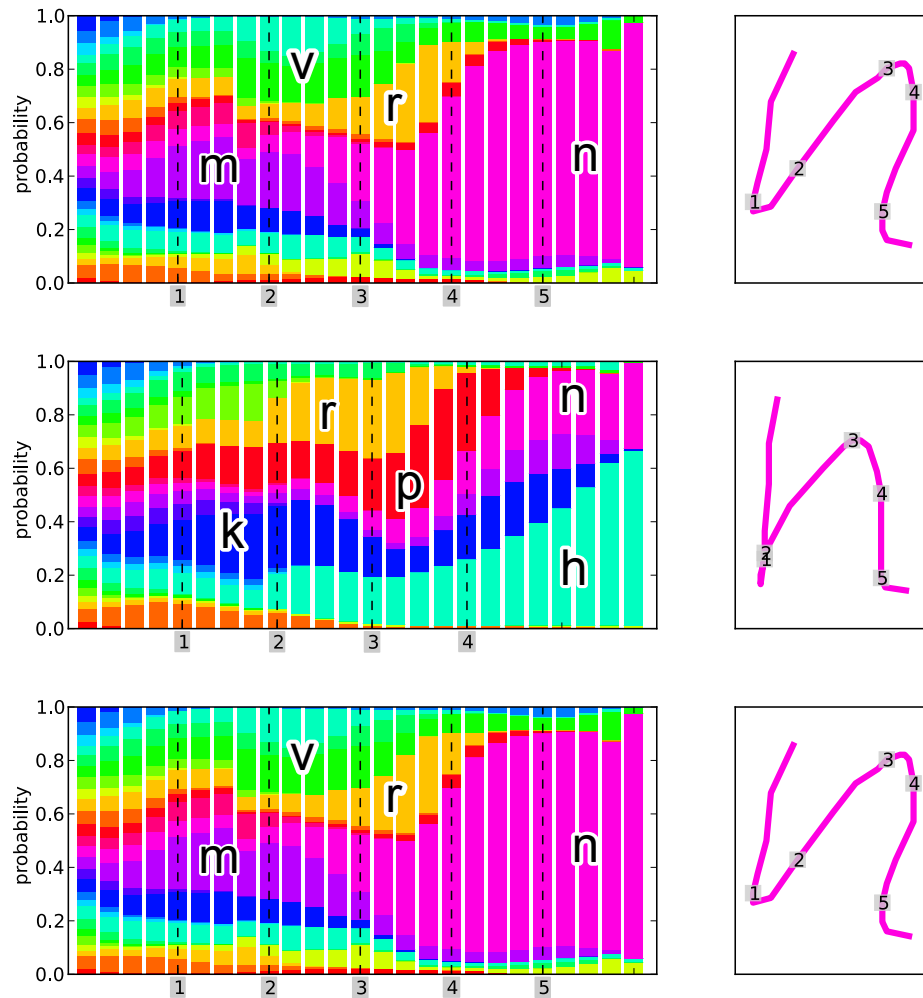


Figure 5.7: Three confusing handwriting examples from a single user. The top example and the bottom example are recognized correctly as an 'n' and an 'h' respectively. The middle example is recognized as an 'h' instead of the true label 'n'.

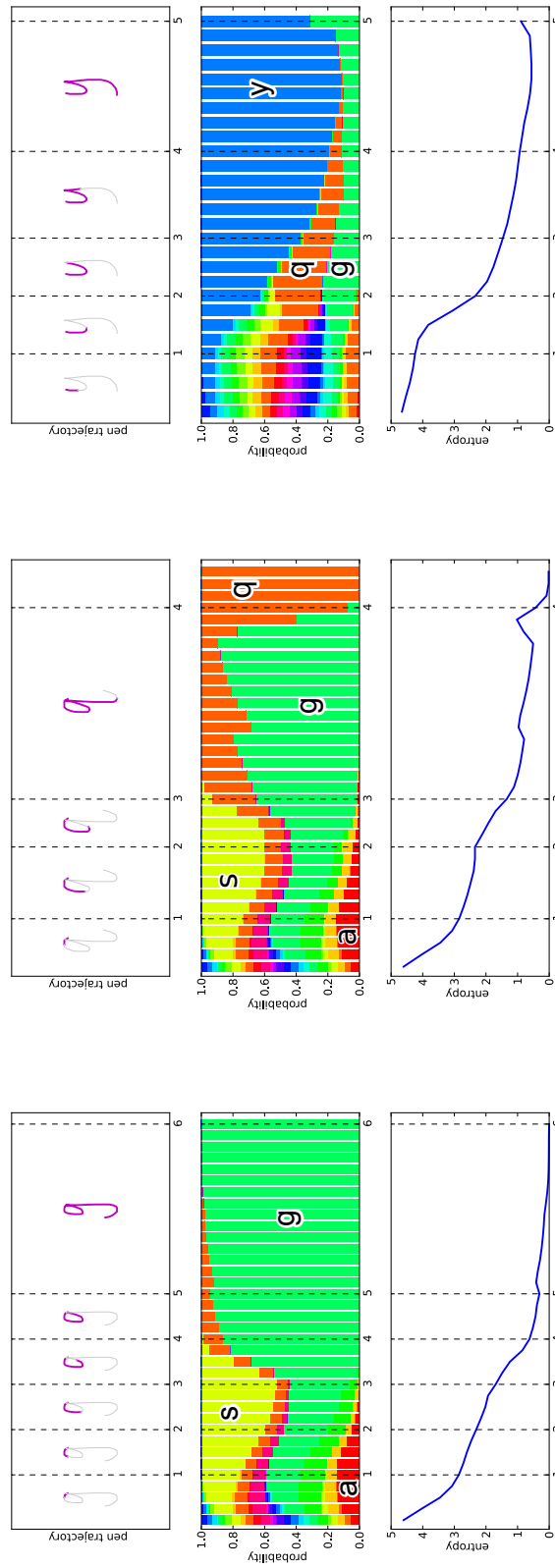


Figure 5.8: Posterior distributions as a function of time. The top row is the partial handwriting trajectories up to each dotted line. The middle row is the likelihood distributions over time where each color corresponds to each label. The bottom row is the entropy over time.

Chapter 6

Improved kNN Rule for Small Training Sets

The traditional k -NN classification rule predicts a label based on the most common label of the k nearest neighbors (the plurality rule). It is known that the plurality rule is optimal when the number of examples tends to infinity. In this chapter we show that the plurality rule is sub-optimal when the number of labels is large and the number of examples is small. We propose a simple k -NN rule that takes into account the labels of all of the neighbors, rather than just the most common label. We present a number of experiments on both synthetic datasets and real-world datasets, including MNIST and SVHN. We show that our new rule can achieve lower error rates compared to the majority rule in many cases.

6.1 Introduction

The k -nearest neighbors (k -NN) algorithm is one of the oldest methods of pattern recognition and machine learning. Given an unlabeled instance s , the algorithm finds the k labeled examples that are closest to s according to some measure of distance or divergence. The algorithm then counts the number of times each of the m possible label appears within this set of k elements and predicts with the label that appears the largest number of times. This is called the “plurality vote”. In the binary label case ($m = 2$), the plurality vote is equal to the majority vote. In this chapter, our focus is on problems

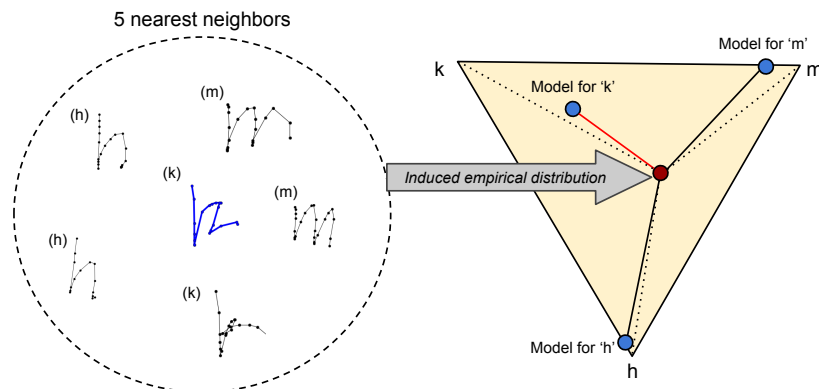


Figure 6.1: Problem with the majority rule when N is small. The five nearest neighbors of an example from handwriting recognition are shown on the left with their true labels shown in parentheses. The majority rule predicts either h or m while the true label is k . However, when the label distribution of the neighborhood is mapped onto the probability simplex shown on the right, a better classification rule is to minimize the KL-divergence (solid lines) between the empirical distribution (red dot) and the typical neighborhood distributions of letter h , m and k (blue dots).

where $m \gg 2$.

Fix and Hodges [FH51] show that for sufficiently large datasets the k -NN rule achieves the Bayes error rate r^* under very mild conditions. More precisely, if n denotes the number of examples which grows to infinity $n \rightarrow \infty$, and we choose k as a function of n so that $k(n) \rightarrow \infty$ and $k(n)/n \rightarrow 0$ then the error rate of the k -NN rule approaches the Bayes optimal error rate. Furthermore, Cover and Hart [CH67] show that, when $k = 1$, the asymptotic error rate of the 1-NN rule is upper bounded by $r^*(2 - \frac{m}{m-1}r^*)$.

However, real-world datasets are always finite and often small. In such cases, the theoretical results give little guidance. Indeed, as we will show, there are good reasons to suggest that rules other than the plurality rule might perform significantly better. The basic intuition is that, in all cases other than the binary case, the plurality vote ignores information present in the counts of labels other than the largest label.

To motivate our approach, consider the following example from handwriting recognition using 5-NN where the distances between examples are computed using a standard elastic matching divergence. Figure 6.1 shows an instance of a handwritten letter “ k ” and its 5 nearest neighbors: two “ h ”s, two “ m ”s and a single “ k ”. The plurality vote will go for either “ h ” or “ m ”, both of which are incorrect. To avoid this mistake, we propose an alternative classification rule. The key is to bring into the model the *average*

neighbor distribution for each letter. In other words, using the labeled training set, we can estimate a *model* for the distribution of labels of the k neighbors for each label. In most cases, the most common neighbor label will be the same as the label of the queried point. However, the distribution will also capture information about the other labels that tend to be in the k -NN set of a point.

In Figure 6.1, we represent the model distributions for the letters “k”, “h” and “n” as points on the 2D triangular simplex. Note that the neighbors for “h” and “m” assign very small probabilities to having the label “k” in their neighborhood, while the letter “k” has a significant probability of having “h” or “m” in its neighborhood. On the same simplex we plot the empirical distribution $(2, 2, 1)$ (indicated by a red dot). By using the Kullback-Leibler divergence, we can recover the fact that the letter is a “k”. This reversal of the classification is based on the following logic: while it is true that there are more instances of “h” and “m” in the 5-NN neighborhood, there exists one instance of “k”. On the other hand, it is common for “h” and “m” to be in the neighborhood set of a “k” while it is very rare for a “k” to be in the neighborhood of an “h” or an “m”. The result is that “k”, even though it is in the minority, is a better explanation of the observed labels.

Our approach is related to work on learning label embeddings [CSM09, BWG10]. The main difference is that our approach is far simpler, does not require any convex optimizations and can be seamlessly integrated into the k -NN framework. Another related work is [BJM01] which introduces a bias term to the likelihood ratio testing which is justified by the difference between the estimated and the true class conditional probability.

This chapter is organized as follows. In Section 6.2, we describe the framework and the notations. In Section 6.3, we describe our approach and justification. In Section 6.4, we present experiments comparing our approach with the traditional k -NN algorithm using both synthetic data and real-world data. Then, we discuss the results in Section 6.5 and conclude the chapter in Section 6.6.

6.2 Background

Let $\mathcal{S} = \{(x_1, y_1), \dots, (x_N, y_N)\}$ be a set of training examples where each instance x_i comes from an example space \mathcal{X} of which the distance between any two examples is measured by $d(\cdot, \cdot)$. Without loss of generality, we assume that each label y_i takes on a value from $\mathcal{Y} = \{1, 2, \dots, m\}$. To simplify the analysis, we assume that the distribution of classes is uniform and the number of examples per class is denoted by n .

Let $\mathcal{N}_k(x)$ denote the neighborhood of size k of an example $x \in \mathcal{X}$ with respect to the distance measure d . The traditional k -NN rule predicts the label of an example x with the majority of the labels in $\mathcal{N}_k(x)$. More formally, given x and $\mathcal{N}_k(x)$, we can define an empirical distribution $\widehat{\mathbf{P}}_{(x, \mathcal{S}, k)}$ such that, for each $i \in \mathcal{Y}$,

$$\widehat{\mathbf{P}}_{(x, \mathcal{S}, k)}(i) = \frac{\#\{\text{occurrences of label } i \text{ in } \mathcal{N}_k(x)\}}{k}$$

The k -NN rule predicts the label \hat{y} such that

$$\hat{y} = \arg \max_{i \in \mathcal{Y}} \widehat{\mathbf{P}}_{(x, \mathcal{S}, k)}(i)$$

For any example $x \in \mathcal{X}$, we can consider the true class distribution of x , denoted by $\mathbf{P}_{(x)}$ which is given by, for each $i \in \mathcal{Y}$,

$$\mathbf{P}_{(x)}(i) = \Pr(Y = i | X = x)$$

Under certain assumptions, it is shown in [FH51] that, for every class label $i \in \mathcal{Y}$,

$$\lim_{\substack{n \rightarrow \infty \\ k \rightarrow \infty \\ k/n \rightarrow 0}} \widehat{\mathbf{P}}_{(x, \mathcal{S}, k)}(i) = \mathbf{P}_{(x)}(i)$$

Therefore, the majority rule is asymptotically optimal. However, in the finite sample scenario, it can be sub-optimal due to the discrepancy between the empirical distribution $\widehat{\mathbf{P}}_{(x, \mathcal{S}, k)}$ and the true distribution $\mathbf{P}_{(x)}$ as demonstrated by the example in Figure 6.1.

6.3 Minimizing KL-Divergence Rule

We propose a new k -NN rule that predicts the class label based on the entire class distribution $\widehat{\mathbf{P}}_{(x, \mathcal{S}, k)}$ instead of just the mode (majority) of $\widehat{\mathbf{P}}_{(x, \mathcal{S}, k)}$. We refer to

Algorithm 5 The MinKL k -NN rule: Training

Initial Assumptions: Training set \mathcal{S} and k **Output:** The center distributions $\widehat{\mathbf{Q}}_j$ for all

- $j \in \mathcal{Y}$
- 1: $\widehat{\mathbf{Q}}_j \leftarrow \vec{0}$ for $j \in \mathcal{Y}$
 - 2: **for** each example $(x, j) \in \mathcal{S}$ **do**
 - 3: $\widehat{\mathbf{Q}}_j \leftarrow \widehat{\mathbf{Q}}_j + \widehat{\mathbf{P}}_{(x, \mathcal{S}, k)}$
 - 4: **end for**
 - 5: $\widehat{\mathbf{Q}}_j \leftarrow \widehat{\mathbf{Q}}_j / |\mathcal{S}_j|$ for all $j \in \mathcal{Y}$
-

Algorithm 6 The MinKL k -NN rule: Prediction

Initial Assumptions: Training set \mathcal{S} ,

A test example x ,

The center distributions $\widehat{\mathbf{Q}}_j$ for all $j \in \mathcal{Y}$ **Output:** Predicted label \hat{y}

- 1: $\hat{y} = \arg \min_{i \in \mathcal{Y}} D_{\text{KL}}(\widehat{\mathbf{P}}_{(x, \mathcal{S}, k)} || \widehat{\mathbf{Q}}_i)$
-

this rule as the minimizing KL-divergence rule (MinKL). Given a training set \mathcal{S} and the neighborhood of size k , we define, for each class j , an empirical center distribution $\widehat{\mathbf{Q}}_{(j, \mathcal{S}, k)}$ as

$$\widehat{\mathbf{Q}}_{(j, \mathcal{S}, k)} = \frac{\sum_{(x, j) \in \mathcal{S}_j} \widehat{\mathbf{P}}_{(x, \mathcal{S}, k)}}{|\mathcal{S}_j|}$$

where $\mathcal{S}_j = \{(x, y) \in \mathcal{S} | y = j\}$ consists of all examples with class label j . To classify a new example x , the empirical class distribution $\widehat{\mathbf{P}}_{(x, \mathcal{S}, k)}$ is compared to each of the center distributions $\widehat{\mathbf{Q}}_{(j, \mathcal{S}, k)}$ with respect to the KL-divergence $D_{\text{KL}}(\widehat{\mathbf{P}}_{(x, \mathcal{S}, k)} || \widehat{\mathbf{Q}}_{(j, \mathcal{S}, k)})$ and the class label that minimizes the distance is then predicted. More formally, the predicted label \hat{y} is given by

$$\hat{y} = \arg \min_{j \in \mathcal{Y}} D_{\text{KL}}(\widehat{\mathbf{P}}_{(x, \mathcal{S}, k)} || \widehat{\mathbf{Q}}_{(j, \mathcal{S}, k)})$$

where the KL-divergence between two discrete distributions \mathbf{p} and \mathbf{q} is defined as

$$D_{\text{KL}}(\mathbf{p} || \mathbf{q}) = \sum_i \mathbf{p}(i) \log \frac{\mathbf{p}(i)}{\mathbf{q}(i)}$$

A summary of the algorithm is given in Algorithm 5 and Algorithm 6.

To analyze our approach in the finite sample setting, we introduce a few more notations. Let $\vec{\mathbf{P}}_{(x, k)}$ denote the expected class distribution of an example x induced by

a neighborhood of size k , which is given by

$$\vec{\mathbf{P}}_{(x,k)} = \mathbf{E}_{\mathcal{S}}[\widehat{\mathbf{P}}_{(x,\mathcal{S},k)}]$$

Similarly, let $\vec{\mathbf{Q}}_{(j,k)}$ denote the expected center distribution for examples of class j defined by

$$\vec{\mathbf{Q}}_{(j,k)} = \mathbf{E}_{\mathcal{S}}[\widehat{\mathbf{Q}}_{(j,\mathcal{S},k)}]$$

Note that the expectation is taken over all possible training sets of size N .

Ideally, the empirical distribution $\widehat{\mathbf{P}}_{(x,\mathcal{S},k)}$ should be compared to the expected center distribution $\vec{\mathbf{Q}}_{(j,k)}$. However, in practice, we use $\widehat{\mathbf{Q}}_{(j,\mathcal{S},k)}$ as an estimate for $\vec{\mathbf{Q}}_{(j,k)}$. This is reasonable because $\widehat{\mathbf{Q}}_{(j,\mathcal{S},k)}$ for each class j is estimated from a relatively large amount of examples in the training set.

To justify our approach, we begin by defining the KL-divergence between a distribution p and a set of distributions E . Then, we prove a lemma to show that an empirical distribution \hat{p} induced by drawing examples from a true distribution $p \in E$ is likely to be “close” to the set E in the KL-divergence sense.

Definition 6.1. *The KL-divergence between any distribution p and a set of distributions E is defined as*

$$D_{\text{KL}}(p||E) \doteq \arg \min_{q \in E} D_{\text{KL}}(p||q) \quad (6.1)$$

Lemma 6.2. *Let E be a set of distributions over some domain \mathcal{X} . For any $p \in E$, let x^n be a sample of size n drawn from p and let \hat{p} be the empirical distribution induced by x^n . Then,*

$$\Pr\{D_{\text{KL}}(\hat{p}||E) > \varepsilon\} \leq (n+1)^{|\mathcal{X}|} e^{-n\varepsilon}$$

Proof. Theorem 11.2.1 in [CT91] states that, for any distribution p and any $\varepsilon > 0$,

$$\Pr\{D_{\text{KL}}(\hat{p}||p) > \varepsilon\} \leq (n+1)^{|\mathcal{X}|} e^{-n\varepsilon}$$

Since $p \in E$, by definition,

$$\Pr\{D_{\text{KL}}(\hat{p}||E) > \varepsilon\} \leq \Pr\{D_{\text{KL}}(\hat{p}||p) > \varepsilon\}$$

So we have

$$\Pr\{D_{\text{KL}}(\hat{p}||E) > \varepsilon\} \leq (n+1)^{|\mathcal{X}|} e^{-n\varepsilon}$$

□

Suppose E_1, E_2, \dots, E_m be sets of distributions such that $\bigcap E_i = \emptyset$. We assume that a sample of size n , x^n is generated by the following process:

1. Class i^* is chosen with probability π_{i^*} .
2. A distribution p is chosen such that $\Pr\{p \in E_{i^*}\} \geq 1 - \delta$
3. x^n is sampled from the distribution p .

Theorem 6.3. *Let E_1, E_2, \dots, E_m be sets of distributions such that $\bigcap E_i = \emptyset$. Suppose x^n is generated according to the above process with respect to the correct class i^* and the true distribution p where $\Pr\{p \in E_{i^*}\} \geq 1 - \delta$. Let \hat{p} denote the empirical distribution induced by x^n . Then,*

$$\begin{aligned} \Pr\{D_{\text{KL}}(\hat{p}||E_{i^*}) > \min_{j \neq i^*} D_{\text{KL}}(\hat{p}||E_j) \mid p \in E_{i^*}\} \\ \leq (m-1)(n+1)^{|\mathcal{X}|} e^{-n\Delta} \end{aligned}$$

where $\Delta \doteq \min_{i,j;j \neq i} \min_{q \in \mathcal{P}} \max(D_{\text{KL}}(q||E_j), D_{\text{KL}}(q||E_i))$

Proof. By applying Lemma 6.2

$$\begin{aligned} \Pr\{D_{\text{KL}}(\hat{p}||E_{i^*}) > \min_{j;j \neq i^*} D_{\text{KL}}(\hat{p}||E_j) \mid p \in E_{i^*}\} \\ \leq \sum_{j;j \neq i^*} \Pr\{D_{\text{KL}}(\hat{p}||E_{i^*}) > D_{\text{KL}}(\hat{p}||E_j) \mid p \in E_{i^*}\} \\ \leq \sum_{j;j \neq i^*} \Pr\{D_{\text{KL}}(\hat{p}||E_{i^*}) > \Delta \mid p \in E_{i^*}\} \\ \leq (m-1)(n+1)^{|\mathcal{X}|} e^{-n\Delta} \end{aligned}$$

□

Using Theorem 6.3, we can justify Algorithm 7 under the assumptions about the data generation process. It is worth noting that Algorithm 7 is not quite the same as Algorithm 6 since, in Algorithm 6, we only compare the KL distance from the empirical distribution \hat{p} to a single center distribution $\hat{\mathbf{Q}}_j$. However, we can show that if each E_j contains only distributions that are α -close to the center $\hat{\mathbf{Q}}_j$, then Algorithm 6 is also justified by applying Lemma 6.5.

Algorithm 7 Theoretical MinKL

Initial Assumptions: The number of classes m , the sets of distributions E_1, E_2, \dots, E_m , and the empirical distribution \hat{p}

Output: Predict i^*

- 1: **for all** $i \in 1, \dots, m$ **do**
 - 2: $p_i \leftarrow \arg \min_{q \in E_i} D_{\text{KL}}(\hat{p} \| q)$
 - 3: **end for**
 - 4: $i^* = \arg \min_i D_{\text{KL}}(\hat{p} \| p_i)$
-

Definition 6.4. For any $\alpha > 0$, a distribution p is said to be α -close to another distribution q if and only if

$$(1 - \alpha)q(x) \leq p(x) \quad \forall x \in \mathcal{X}$$

Lemma 6.5. If every $q \in E_j$ is α -close to $\hat{\mathbf{Q}}_j$, then

$$D_{\text{KL}}(p \| E_j) \leq D_{\text{KL}}(p \| \hat{\mathbf{Q}}_j) + \log \frac{1}{1 - \alpha}$$

for any distribution p

Proof.

$$\begin{aligned}
D_{\text{KL}}(p \| E_j) &= \min_{q \in E_j} D_{\text{KL}}(p \| q) \\
&= \min_{q \in E_j} \sum_x p(x) \log \frac{p(x)}{q(x)} \\
&\leq \min_{q \in E_j} \sum_x p(x) \log \frac{p(x)}{(1 - \alpha)\hat{\mathbf{Q}}_j(x)} \\
&\leq \sum_x p(x) \log \frac{p(x)}{(1 - \alpha)\hat{\mathbf{Q}}_j(x)} \\
&\leq D_{\text{KL}}(p \| \hat{\mathbf{Q}}_j) + \log \frac{1}{1 - \alpha}
\end{aligned}$$

□

6.4 Experiments

In this section, we describe experiments we have performed with both synthetic data and real-world data. For each dataset, we compare the error rates of the k -NN with

Table 6.1: Summary of the datasets used in our experiments.

Dataset	No. of classes	No. of train ex.	No. of test ex.
SYN-1	10	up to 1600	10000
SYN-2	64	up to 6400	6400
SYN-3	10	up to 1600	10000
uRight	26	9945	-
MNIST	10	60000	10000
SVHN	10	73257	26032

the minimizing KL-divergence rule (MinKL) to those of the k -NN with the majority rule (Majority) under various conditions. A summary of the datasets is given in Table 6.1.

Synthetic data

We performed 3 experiments using synthetic data that can be described as follows. Each example x is a point inside a wrap-around d -dimensional hypercube of size b , or $x \in [0, b - 1]^d$. The instances of each class are generated by a normal distribution with mean located at each integer lattice point of the hypercube and a covariance matrix $\sigma \mathbf{I}_d$. Thus, the total number of classes is b^d . The distribution of the classes in each dataset is uniform. In Figure 6.2, the generating distributions of each dataset are shown in the left-most column. The Manhattan distance (L_1 norm) is used for measuring the distance between examples.

In our first experiment, we generated a dataset called **SYN-1** using the following parameters: $b = 10, d = 1$ and $\sigma = 1.5$. **SYN-1** was intended to mimic the situation described in Figure 6.1. The number of classes in **SYN-1** is 10. In the second experiment, we generated another dataset called **SYN-2** using the following parameters: $b = 4, d = 3$ and $\sigma = 0.4$. **SYN-2** has a very similar structure to **SYN-1** but it is more complex with the total of 64 classes. In our third experiment, we generated yet another dataset called **SYN-3**. Similar to **SYN-1**, each instance of **SYN-3** is one-dimensional. However, the generating distribution for class i is a mixture of two normal distributions centered at i and $i + 3$ and the mixing coefficient is 0.8 and 0.2 respectively. **SYN-3** is intended for simulating when $\delta_k > 0$.

In Figure 6.2, we compare the error rates of MinKL and Majority using different

n and k for each dataset. For each n , we ran both MinKL and Majority for k ranged. The center column of Figure 6.2 shows the error rates for different k when n is fixed at 20 per classes for each dataset. Then, for each n , the best error rate of both MinKL and Majority over k are shown in the right-most column of Figure 6.2. The error rates of both MinKL and Majority converge to the Bayes error as n increases. In **SYN-1** and **SYN-2**, MinKL converges faster than Majority and is able to attain lower error rates especially when n is small. However, in **SYN-3**, MinKL has higher average error rates than Majority for when n is small.

uRight

The uRight dataset contains handwriting trajectories of the 26 lowercase English characters. We collected the handwriting data from 15 different users writing isolated lowercase English characters on a touch screen of a mobile phone with their fingers. Each example is a sequence of (x, y, t) where x and y are the (x, y) -coordinates and t is the timestamp of each sample point. Figure 6.3 shows some examples of the handwriting trajectories. There are 9945 examples in the dataset and the distribution of the class labels is fairly uniform. The similarity between two examples is measured by the dynamic time warping (DTW) distance [BB04].

Using $k = 5$, the average error rates of MinKL and Majority for each user are summarized in Figure 6.4. According to the paired t-test, the average error rate of MinKL (3.76%) is significantly smaller than the average error rate of Majority (5.86%) with p -value < 0.001 . Figure 6.5 displays some of the examples that were misclassified by Majority but correctly classified by MinKL.

MNIST

The MNIST dataset [LBBH98] contains images of handwritten digits. Each example is a 28x28 grayscale image. There are 60000 training examples and 10000 test examples included in the dataset. We preprocessed the data by de-skewing and down-sampling the images. After the preprocessing, we ran PCA on the training data. The feature vector of each example corresponds to the coefficients of the first 100 PCA com-

ponents. The Euclidean distance is used as the similarity measure in the neighborhood calculation.

The test error rates we obtained from our experiment are comparable to what reported in [LBBH98]. The performance of both MinKL and Majority are very similar for this dataset. The lowest error rate of 1.89% for Majority and 1.90% for MinKL was obtained when $k = 5$. Figure 6.7 shows the test error rates of both MinKL and Majority obtained using different k .

SVHN

The SVHN dataset [NWC⁺11] contains images of digits taken from the Google street view data. It is considered a harder dataset than MNIST due a higher degree of variations. Each example in SVHN is a 32x32 RGB image. There are 73257 training examples and 26032 test examples included in the dataset. We computed, for each example, the HOG features [DT05] using the block size of 4x4 with 8 orientations per block. The Manhattan distance is used as the similarity measure in the neighborhood calculation.

In [NWC⁺11], the test error rate for HOG features combined with an SVM is reported to be around 15%. In our experiment, the test error rates of both MinKL and Majority are between 16% to 17% with MinKL performing slightly better than Majority at every $k > 1$. Figure 6.7 shows the test error rates of both MinKL and Majority obtained using different k .

6.5 Discussion

In our experiments with **SYN-1** and **SYN-2**, we observed that MinKL performs significantly better than Majority when n is small. This result also confirms our intuition we have on the example in Figure 6.1. Our explanation for this boost in performance is the fact that, for small n (implied a small k), the majority rule is prone to error because the prediction is based on solely the majority of the empirical class distribution $\hat{\mathbf{P}}_{(x, \mathcal{S}, k)}$ induced from a relatively small k ; while the MinKL rule makes the prediction based on the entire class distribution.

From our analysis in Section 6.3, we show that our approach will perform optimally when the training set has $\delta_k = 0$. However, when δ_k is small relative to the distance between the center distributions of different classes, we expect our approach to perform reasonably well. In **SYN-3**, we deliberately designed the dataset such that its δ_k is quite large. As expected, the error rates for our approach are inferior to those of the majority rule even when n is small. A workaround is to increase the representation power of the center distribution by allowing multiple centers per class.

The performance gap between the MinKL rule and the majority rule for MNIST is very small, especially for small k . This is due to the fact that the center distributions for the MNIST dataset are very close to the Dirac delta function in which case the MinKL reduces to the majority rule. Figure 6.6 depicts the center distributions in each row of the matrix. For MNIST, the mass is heavily concentrated on the diagonal.

Another factor that plays a role in the performance of MinKL is the distances between the center distributions. If they are far away from each other, then we expect our approach to work well. This effect is also demonstrated in Figure 6.6. The rows in the uRight matrix is more distinct than the rows in the SVHN matrix. The improvement we obtained on the uRight dataset is significantly larger than what we obtained on the SVHN dataset.

In practice, other divergences might work better than the KL-divergence. The KL-divergence is considered a special case of a more general divergence function called Alpha-divergence [CA10], which is given by

$$D_A^{(\alpha)}(p||q) = \frac{1}{\alpha(\alpha-1)} \left(\sum_1^n p_i^\alpha q_i^{1-\alpha} - 1 \right), \alpha \in \mathbb{R} \setminus \{0, 1\}$$

The KL-divergence can be expressed as $D_{\text{KL}}(p||q) = \lim_{\alpha \rightarrow 1} D_A^{(\alpha)}(p||q)$. For the uRight dataset, we were able to obtain even lower error rate by using Alpha-divergence with $\alpha = 2$.

Technically, the minimizing KL-divergence idea can be applied to other classification algorithms as well. The k -NN algorithm is very computational expensive in classifying a new example. In some applications, it is important to be able to classify new examples quickly. A simple modification to the k -NN algorithm that significantly reduces the classification time is to keep only a small number of representatives per class

and discard the rest of the examples. This algorithm is called the k nearest-centroid algorithm (k -NC) where only the k -centroids are kept as the class representatives. In the k -NC, the class distribution \mathbf{P}_x can be estimated by

$$\hat{\mathbf{P}}_x(j) = \frac{e^{d(x,C(j))}}{\sum_i e^{d(x,C(i))}}$$

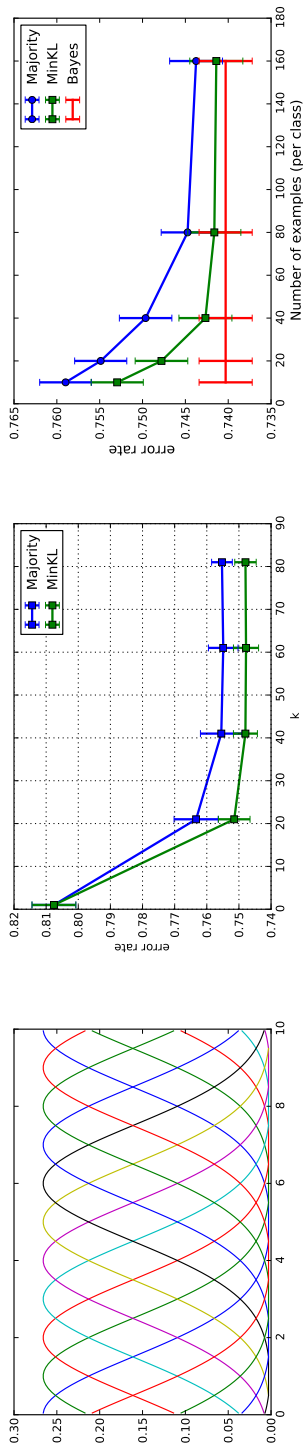
Then we can apply MinKL rule to the class posterior computed above.

6.6 Conclusions

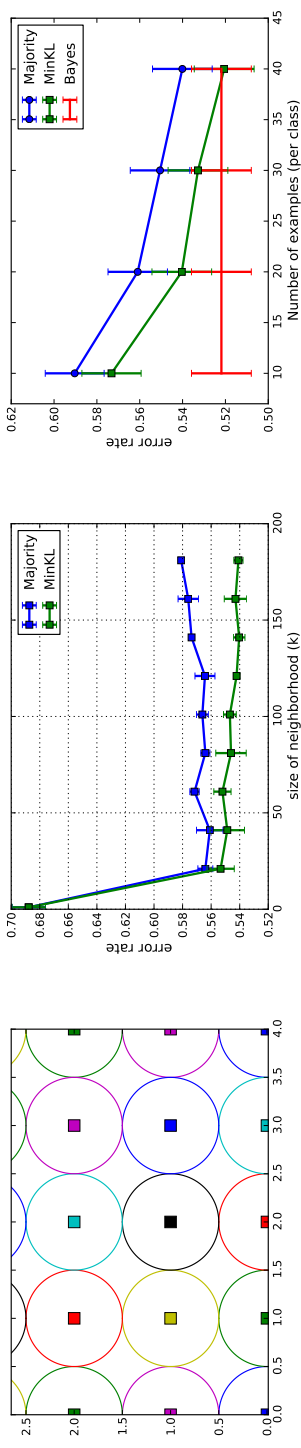
We proposed a simple k -NN rule that predicts based on the entire empirical class distribution rather than the majority in the neighborhood. The algorithm can be described as follows. Given a training set, we estimate the center distribution for each class. To classify a new example, we measure the KL-divergence between the empirical distribution induced by the neighborhood and the center distribution of each class. The class that minimized the KL-divergence is then predicted. In a sense, our approach is a simple method for leveraging the class information in the label space. We justified our approach in the finite sample setting under a certain assumption about the data. Finally, we show experimental results comparing the error rates of our approach to the majority rule. We found that our approach managed to outperform the majority rule in many cases.

Acknowledgements

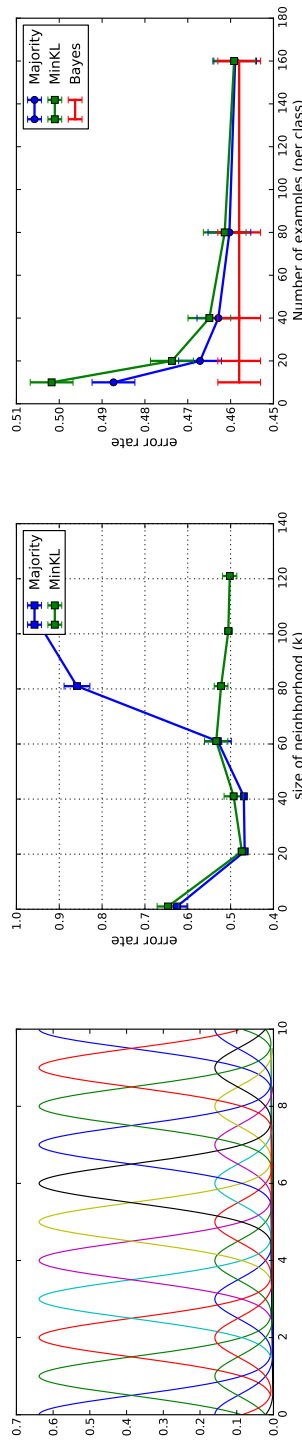
This chapter is based on unpublished work that is currently in submission as of the writing of this thesis. It is joint work with Yoav Freund. The dissertation author is the primary investigator and author of this work.



(a) SYN-1: (left) Data models, (center) error rate vs. k while n fixed, (right) error rate vs. n while k fixed



(b) SYN-2: (left) Data models, (center) error rate vs. k while n fixed, (right) error rate vs. n while k fixed



(c) SYN-3: (left) Data models, (center) error rate vs. k while n fixed, (right) error rate vs. n while k fixed

Figure 6.2: Results from the synthetic data experiment.

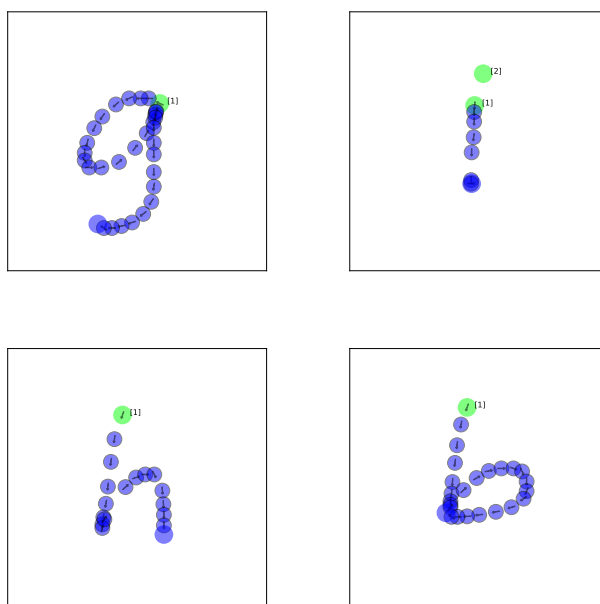


Figure 6.3: Handwriting trajectories from the uRight handwriting dataset.

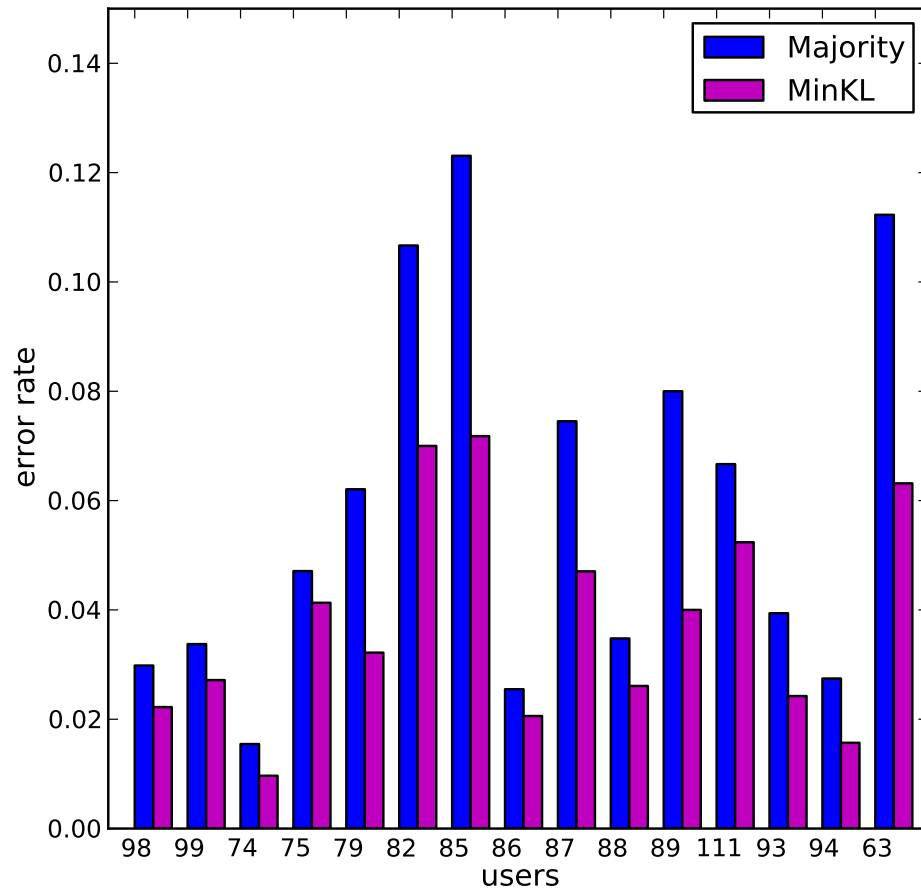


Figure 6.4: Average error rates of MinKL and Majority for each user.

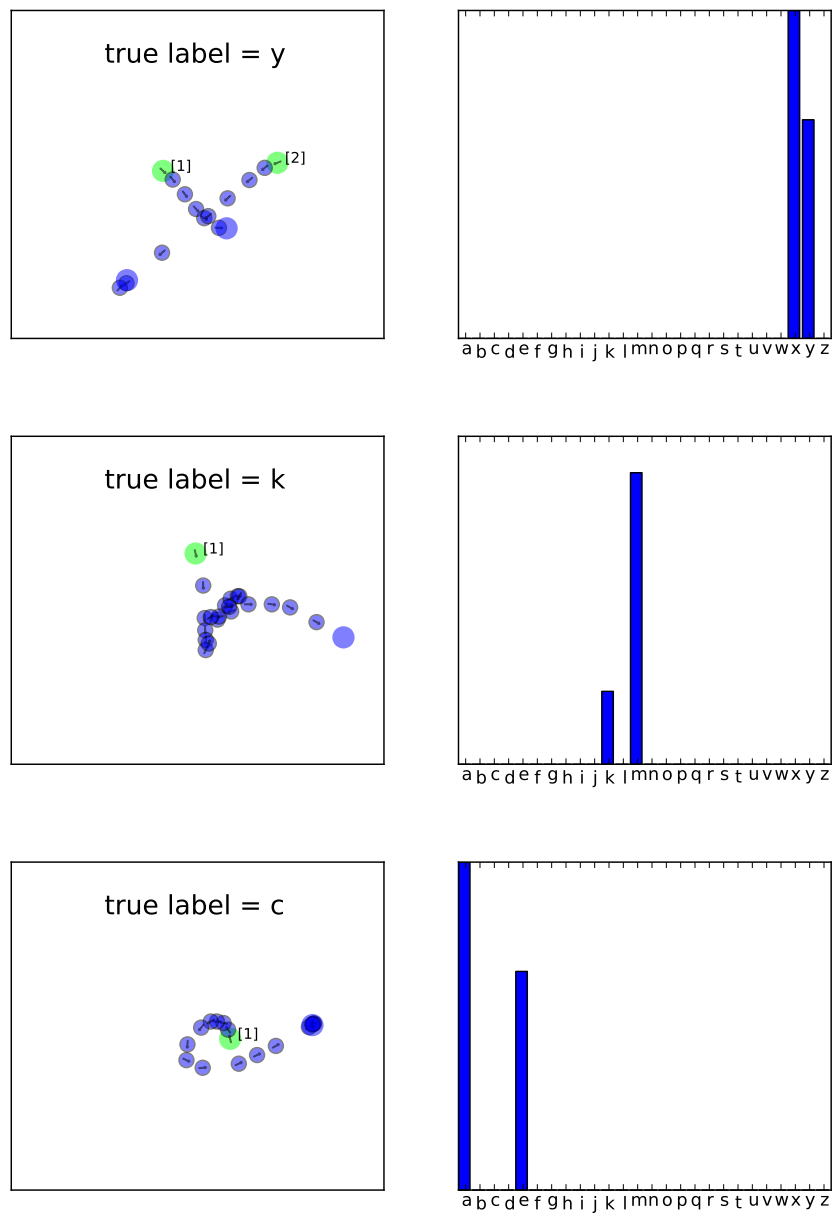


Figure 6.5: Examples classified incorrectly by the majority rule but correctly by the MinKL rule. Each handwriting trajectory is shown on the left and the corresponding empirical distribution induced by its 5 neighbors is shown on the right.

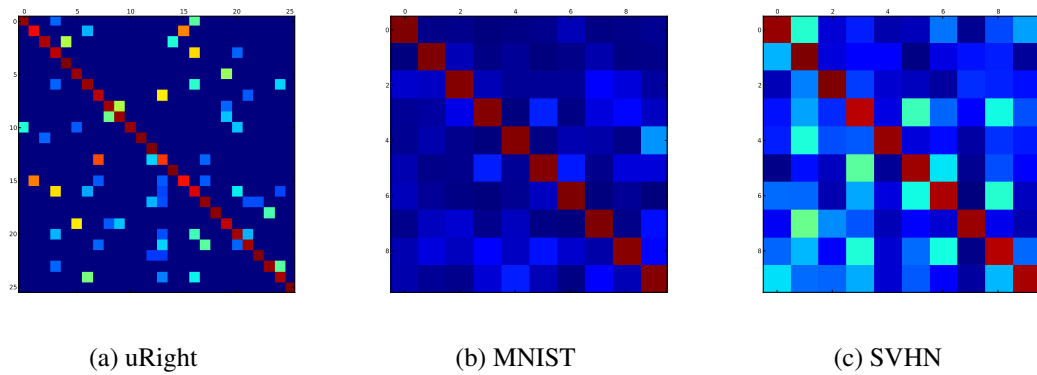
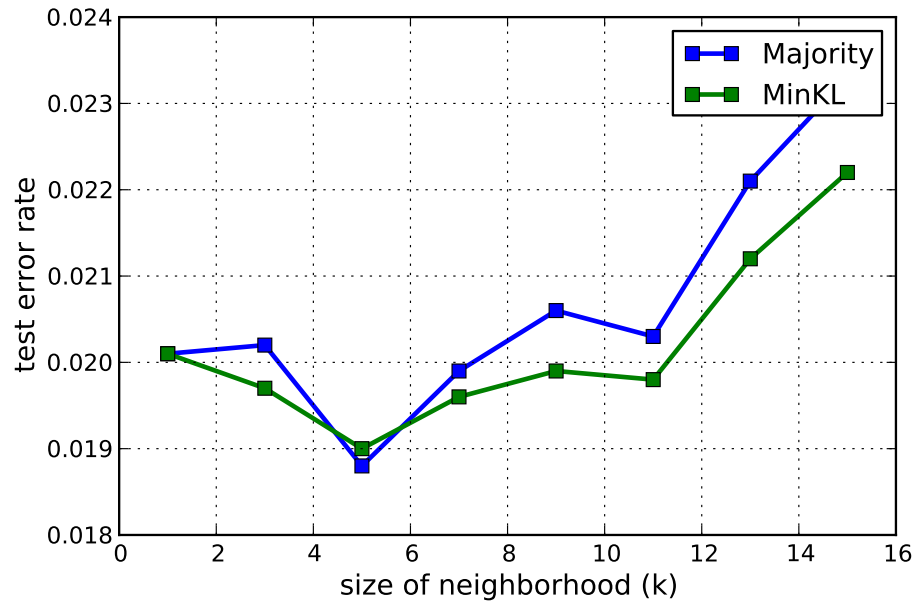
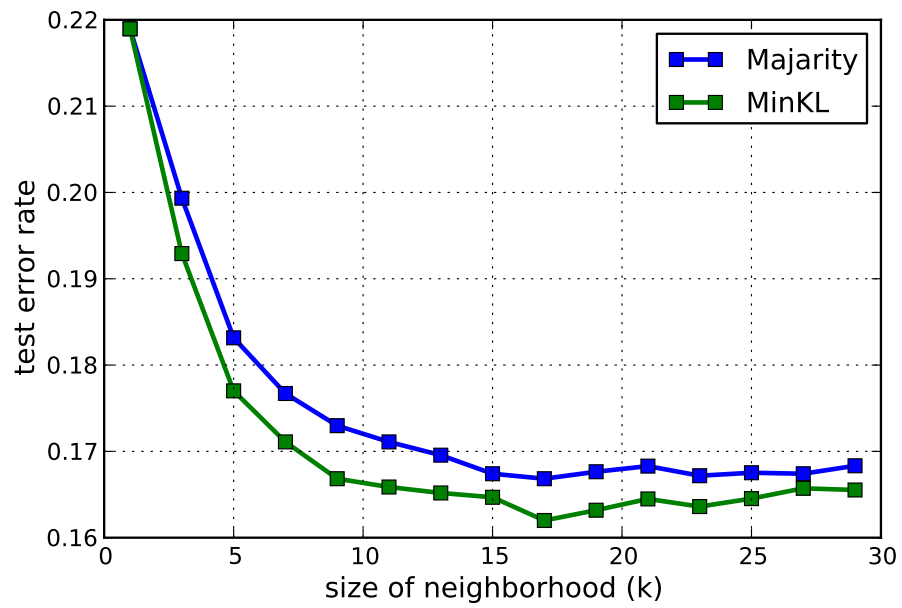


Figure 6.6: Visualization of the empirical center distributions. Each row in each matrix corresponds to the center distribution for each class.



(a) MNIST



(b) SVHN

Figure 6.7: MNIST and SVHN results

Chapter 7

Conclusions and Recommendations

The problem of intent inference has manifested itself in all classes of input interfaces. The difficulty of the problem quickly increases when the interface has to interpret ambiguous signals such as pen trajectories, video streams, or speech. Our machine learning-based approach has been shown to be effective on our systems for improving robustness and efficiency for the task and can be applied to other recognition systems in the future.

In Chapter 2 we discussed the problem of user engagement in the context of the Automatic Cameraman (TAC). The challenge was to identify the user engagement in real-time. Our solution was to use machine learning methods to process and combine multiple input modalities to detect and track the user. Then, we implemented a simple hand interaction to determine the engagement. Using this method, TAC was able to engage with the users autonomously for days without any human intervention showing the robustness against various conditions of the room.

As the production cost for advanced sensing technology such as high-speed video cameras, MEMS microphones, or 3D depth sensors decreases, the idea of combining multiple input modalities becomes more and more important. The signals produced by a single modality are easily corrupted by noise, especially when the signals are complex and high-dimensional. By combining multiple modalities, we essentially introduce redundancy and effectively reduce noise, allowing for more accurate interpretation of the event.

Another advantage of our method is that the amount of calibration needed for the

system is minimal as most of the parameters are tuned continuously and automatically from observations. Parameter tuning is a tedious task that usually requires several trials and errors. Additionally, a slight modification to the system could invalidate the current setting of parameters and require another round of the calibration. Through machine learning, TAC is able to adapt to changes in the environment and only requires minimal amount of calibration.

It is important for any interactive system to react quickly to the user. The common consensus shows that the reaction time should be below 100ms in order for the interaction to feel natural to a human user. On TAC, the low-level visual feature calculation accounts for a large portion of the CPU time, compromising the overall responsiveness of the entire system. To mitigate the problem, some of the computations were moved to a dedicated implementation on a Field-programmable gate array (FPGA). The study of FPGA was beyond the scope of this dissertation, but it definitely deserves a more detailed investigation. We believe that FPGA-based solution will be a viable option for future interactive systems.

There are plenty of rooms for improvements for TAC. As of current, TAC only supports two commands: start and stop recording. While our method works well for the two commands, it is interesting to see how well the system will perform when more commands are added. The commands could be either voice-based commands or gesture-based commands. As the system supports more commands, it is possible to introduce other tasks to TAC such as gaming or browsing the web.

In Chapter 5 we discussed an approach to improve the input rate of a handwriting recognition system based on the idea of co-adaptation. Recall that co-adaptation is referred to situations where both the human user and the computer system adapt to each other in parallel. Based on our study, we have shown that both human adaptation and computer adaptation have significant impact on the overall information transfer rate in the context of handwriting recognition. In other words, both sides of adaptation should be considered in order to achieve an efficient handwriting recognition system.

Our study was done deliberately on mobile devices where handwriting recognition is a promising yet underutilized input method. We were motivated by the inefficiency of the mainstream soft-keyboard where each letter is represented by a small

button on the touch screen. The situation is worsen when the user needs to enter text in multiple languages. Typically, to enter text in multiple languages, the user needs to switch between different keyboard layouts and each layout has its own set of letters and placement. Handwriting recognition is a promising approach to the multilingual problem. However, its downside is due to its limited information transfer rate. Our study suggested that machine adaptation works best when matched with human adaptation and that co-adaption could be a contributing factor towards a more efficient handwriting recognition system.

The concept of co-adaptation is not only limited to handwriting recognition. It can be applied to other recognition systems such as voice recognition and gesture recognition as well. While machine adaptation is already well-utilized in many recognition systems, the notion of user adaptation has not been explored enough. Most recognition systems simply do not provide enough feedback to the users when they make mistakes. This often leads to confusion, an unpleasant user experience and an inefficient communication due to repetitive errors. Although we were able to suggest that giving feedback to users is important, the question of how and when the feedback should be delivered still remains for future studies.

Bibliography

- [AMGC02] M S Arulampalam, S Maskell, N Gordon, and T Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *Signal Processing, IEEE Transactions on*, 50(2):174–188, February 2002.
- [BB04] C Bahlmann and H Burkhardt. The writer independent online handwriting recognition system frog on hand and cluster generative statistical dynamic time warping, 2004.
- [BBZ07] Maria-Florina Balcan, Andrei Broder, and Tong Zhang. Margin Based Active Learning. In Nader H Bshouty and Claudio Gentile, editors, *Learning Theory 20th Annual Conference on Learning Theory COLT 2007 San Diego CA USA June 1315 2007 Proceedings*, volume 4539 of *Lecture Notes in Artificial Intelligence*, pages 35–50. Springer Berlin Heidelberg, 2007.
- [BHB02] Claus Bahlmann, Bernard Haasdonk, and Hans Burkhardt. On-Line Handwriting Recognition with Support Vector Machines - A Kernel Approach. In *Proceedings of the Eighth International Workshop on Frontiers in Handwriting Recognition (IWFHR '02)*, pages 49–54. IEEE Computer Society, 2002.
- [Bil66] Edward A Bilodeau. *Acquisition of skill*. Academic Press, New York, NY, USA, 1966.
- [Bil97] J Bilmes. A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical report, ICSI, 1997.
- [BJM01] J. Bilmes, G. Ji, and M. Meila. Intransitive likelihood-ratio classifiers. *Advances in Neural Information Processing Systems*, pages 0–4, 2001.
- [BK13] Jakramate Bootkrajang and A Kabán. Boosting in the presence of label noise. *UAI*, 2013.
- [BL13] K Bache and M Lichman. {UCI} Machine Learning Repository, 2013.

- [BM08] Olivier Bau and Wendy E Mackay. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '08, pages 37–46, New York, NY, USA, 2008. ACM.
- [BWG10] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. *Advances in Neural Information Processing Systems*, 23(1):1–10, 2010.
- [CA10] A. Cichocki and S. Amari. Families of Alpha- Beta- and Gamma-Divergences: Flexible and Robust Measures of Similarities. *Entropy*, 12(6):1532–1568, June 2010.
- [CAL94] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.
- [CBCK10] Junguk Cho, Bridget Benson, Sunsern Cheamanukul, and Ryan Kastner. Increased performance of FPGA-based color classification system. In *Proceedings - IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2010*, pages 29–32, 2010.
- [CBL06] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA, 2006.
- [CFH09] Kamalika Chaudhuri, Yoav Freund, and Daniel Hsu. Tracking using explanation-based modeling. *CoRR*, abs/0903.2, March 2009.
- [CFH10] Kamalika Chaudhuri, Y Freund, and Daniel Hsu. An Online Learning-based Framework for Tracking. In *UAI*, 2010.
- [CH67] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.
- [CJ02] S D Connell and A K Jain. Writer adaptation for online handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(3):329–346, 2002.
- [CL11] C.-C. Chang and C.-J. Lin. {LIBSVM}: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1—27:27, 2011.
- [CSM09] M. Collins and N. Singh-Miller. Learning label embeddings for nearest-neighbor multi-class classification with an application to speech recognition. *Advances in Neural Information Processing Systems*, pages 1–9, 2009.

- [CST00] N Cristianini and J Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, New York, NY, USA, 2000.
- [CT91] Thomas M Cover and Joy A Thomas. *Elements of Information Theory*, volume 6 of *Wiley Series in Telecommunications*. Wiley, 1991.
- [CV95] C Cortes and V Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- [Die00] Thomas G Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*, 40(2):139–157, 2000.
- [DT05] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1, 2005.
- [Ett10] Evan Ettinger. *Audio Localization in the Automatic Cameraman*. PhD thesis, University of California, San Diego, 2010.
- [FH51] E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination. *USAF School of Aviation Medicine, Randolph Field, Texas, Project 21-49-004, Report 4*, 1951.
- [FHM95] Clive Frankish, Richard Hull, and Pam Morgan. Recognition accuracy and user acceptance of pen interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '95*, pages 503–510, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.
- [FHT98] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive Logistic Regression: a Statistical View of Boosting. *Annals of Statistics*, 28:2000, 1998.
- [Fit54] P M Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954.
- [FM99] Yoav Freund and Llew Mason. The Alternating Decision Tree Learning Algorithm. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 124–133, 1999.
- [Fre95] Yoav Freund. Boosting a weak learning algorithm by majority. *Inf. Comput.*, 121(2):256–285, 1995.
- [Fre01] Yoav Freund. An Adaptive Version of the Boost by Majority Algorithm. *Mach. Learn.*, 43(3):293–318, 2001.

- [Fre09] Yoav Freund. A more robust boosting algorithm. *arXiv:0905.2138 [stat.ML]*, 2009.
- [FS96] Yoav Freund and Robert E Schapire. Experiments with a New Boosting Algorithm. In *Proceedings of the 13th International conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.
- [FS99] Yoav Freund and Robert E Schapire. A Short introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, September 1999.
- [GR93] David Goldberg and Cate Richardson. Touch-typing with a stylus. In *Proceedings of the SIGCHI conference on Human factors in computing systems CHI 93, CHI '93*, pages 80–87. ACM Press, 1993.
- [GWM⁺03] Matthew Garrett, David Ward, Iain Murray, Phil Cowans, and David Mackay. Implementation of Dasher, an information efficient input mechanism. *Nature*, pages 1–6, 2003.
- [Hö0] K Höök. Steps to take before intelligent user interfaces become real. *Interacting with computers*, 12:409–426, 2000.
- [HBT96] J Hu, M K Brown, and W Turin. HMM based online handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(10):1039–1045, October 1996.
- [HL02] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, March 2002.
- [JH99] TS S Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 487–493, Cambridge, MA, USA, 1999. MIT Press.
- [JL05] Alejandro Jaimes and Jianyi Liu. Hotspot Components for Gesture-Based Interaction. In *INTERACT 2005*, pages 1062–1066, 2005.
- [JMW00] S Jaeger, S Manke, and A Waibel. Npen++: An On-Line Handwriting Recognition System. In *in 7th International Workshop on Frontiers in Handwriting Recognition*, pages 249–260, 2000.
- [KAB08] K Kumara, R Agrawal, and C Bhattacharyya. A large margin approach for writer independent online handwriting classification. *Pattern Recognition Letters*, 29(7):933–937, 2008.

- [KC06] Wolf Kienzle and K Chellapilla. Personalized handwriting recognition via biased regularization. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*, number Section 6, pages 457–464, Pittsburgh, Pennsylvania, 2006.
- [KK09] A Kalai and V Kanade. Potential-Based Agnostic Boosting. *NIPS*, pages 1–11, 2009.
- [LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.
- [LD09] Brian Y. Lim and Anind K. Dey. Assessing demand for intelligibility in context-aware applications. *Proceedings of the 11th international conference on Ubiquitous computing - Ubicomp '09*, page 195, 2009.
- [LS08] Philip M. Long and Rocco a. Servedio. Random classification noise defeats all convex potential boosters. *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 608–615, 2008.
- [Mae94] P Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 1994.
- [MGDV93] N Matic, I Guyon, J Denker, and V Vapnik. Writer adaptation for on-line handwritten character recognition. In *Proceedings of the Second International Conference on Document Analysis and Recognition (ICDAR '93)*, pages 187–191. IEEE, 1993.
- [MO97] Richard Maclin and David Opitz. An Empirical Evaluation of Bagging and Boosting. In *In Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 546–551, 1997.
- [MZ97] I Scott Mackenzie and Shawn X Zhang. The immediate usability of graffiti. In *Proceedings of Graphics Interface '97*, pages 129–137, 1997.
- [NR81] A Newell and P S Rosenbloom. Mechanisms of skill acquisition and the law of practice. In J R Anderson, editor, *Cognitive skills and their acquisition*, volume 6 of *Cognitive skills and their acquisition*, chapter 1, pages 1–55. Erlbaum, 1981.
- [NWC⁺11] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, pages 1–9, 2011.
- [PS00] R Plamondon and S N Srihari. On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1):63–84, January 2000.

- [RJ93] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition*, volume 103 of *Prentice Hall signal processing series*. Prentice Hall, 1993.
- [SF12] Robert E Schapire and Yoav Freund. *Boosting: Foundations and Algorithms*. The MIT Press, 2012.
- [SFBL98] Robert E Schapire, Yoav Freund, Peter Bartlett, and W S Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.
- [Sha48] C E Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(July 1928):379–423, 1948.
- [SK87] L Sirovich and M Kirby. Low-dimensional procedure for the characterization of human faces. *J. Opt. Soc. Am. A*, 4(3):519–524, 1987.
- [TAKM04] Koji Tsuda, Shotaro Akaho, Motoaki Kawanabe, and Klaus-Robert Müller. Asymptotic properties of the Fisher kernel. *Neural computation*, 16(1):115–37, January 2004.
- [TG11] Thomas Ploetz and Gernot A Fink. *Markov Models for Handwriting Recognition*. SpringerBriefs in Computer Science. Springer, 2011.
- [TP91] M A Turk and A P Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.
- [vdM11] L van der Maaten. Learning Discriminative Fisher Kernels. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML '11)*, pages 217–224, New York, NY, USA, June 2011. ACM.
- [VJ01] Paul Viola and Michael Jones. Robust Real-time Object Detection. In *International Journal of Computer Vision*, 2001.
- [YWL98] L S Yaeger, B J Webb, and R F Lyon. Combining Neural Networks and Context-Driven Search for On-Line, Printed Handwriting Recognition in the Newton. In *Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop*, pages 275–298, London, UK, 1998. Springer-Verlag.