Some Applications of Machine Learning in Edge Computing, Wireless and Mobile Systems

by

Anantharaghavan Sridhar

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

Master of Science

(Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN–MADISON

2016

The dissertation is approved by the following members of the Final Committee:
Suman Banerjee, Professor, Computer Sciences

I certify that I have read this thesis and that, in my opinion, it is fully adequate in scope and quality as a thesis for the degree of Masters of Science.

Suman Banerjee

Feb 24, 2017

Professor Suman Banerjee
Department of Computer Science
University of Wisconsin-Madison

Dedicated to the resolute scholars who pick up this thesis from the dusty shelves of Memorial Library.

ACKNOWLEDGMENTS

Every one who is seriously involved in the pursuit of science becomes convinced that a spirit is manifest in the laws of the Universe-a spirit vastly superior to that of man, and one in the face of which we with our modest powers must feel humble.

— Albert Einstein

First and foremost, I would like to thank my advisor Professor Suman Banerjee for being an incredible mentor during my time at the University of Wisconsin-Madison. His vision and guidance have helped shape my research skills and have been the strongest force in showing me the wonders of academic life.

Next, I would like to thank my research mentor, Neil Klingensmith, without whose guidance I wouldn't have learnt half as much as I have. I cannot imagine a better grad school experience without the endless brainstorming discussions, support and encouragement from Neil.

Thank you to the Computer Sciences and Electrical Engineering departments at the University of Wisconsin-Madison for providing a wonderful research atmosphere.

Finally, thank you to my family, friends and loved ones. Thank you to my friends Bharadhwaj, Prashanth, Shruthi, Abisheik, Swami and many others for putting up with my quirkiness through 2.5 years of grad school. Thank you to Aparna for helping me find a balance between work and life. Thank you to my Mom and Dad for always having my back and giving me the confidence to pursue my interests.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Recent advancement in Deep Learning for Image Classification and Perception tasks have contributed to incredible advancements in consumer technology. Devices such as Google Home, projects such as self-driving cars are now closer to reality than ever, thanks to the capacity of deep learning algorithms and research. However, the systems research community has been slower to catch up with the wave of deep learning, in the areas of edge computing and wireless systems.

The objective of this thesis is to expose the reader to some possible applications and ideas for applying machine learning and deep learning techniques in systems research. Particularly, the focus of this thesis is on edge computing, wireless and mobile systems where I believe there is large scope for defining new applications for machine learning research.

1 INTRODUCTION

The objective of this thesis is to present some applications of machine learning and deep learning in systems research. Specifically, this thesis is organized as follows.

Chapter 2 presents an application of Machine Learning in edge computing systems research. The Edge Computing application here deals with improving the performance of water softeners and improving water utility efficiency. The approach taken is to engineer appropriate features so that water hardness classification and forecasting is reduced to a linear classification task that would be computationally light to be programmed on edge computing systems (wireless routers etc.).

Chapter 3 presents an application of Machine Learning in wireless systems research. The wireless systems research application here deals with estimating wireless spectrum usage from sparse measurements across a geographic area (a small city in this case). The approach taken is to use collaborative filtering and estimate the missing spectrum measurements across time at a specific location. In other words, the approach attempts to characterize the time dynamics of wireless spectrum usage.

Chapter 4 presents an application of Deep Learning in mobile systems research. The mobile systems research application here deals with preserving user-privacy in developing voice-enabled mobile applications. The approach taken is to investigate lightweight audio obfuscation techniques that maintain good keyphrase recognition performance.

Chapter 5 presents conclusions and directions for future research.

2 WATER HARDNESS DETECTION IN BUILDING WATER SUPPLY

2.1 Introduction

Domestic and Industrial water supply contain dissolved minerals of varying concentration. The "hardness" of water [5] is used to refer to a certain subset of dissolved minerals, specifically Calcium and Magnesium. In domestic water supply, the dissolved minerals contributing to hardness affect the longevity of appliances [9] while simulatenously increasing health risks to the occupants [33] [24]. In industrial water supply, especially medical research labs, mandate extremely low hardness levels. Consequently, water softening solutions are sought to minimize the ill effects of water hardness.

A typical solution to the water hardness levels is to install water softeners in the consuming units (residential or industrial units). Water Softeners are a primary source of Sodium and Chloride ion pollution in fresh water sources. The pollution arises as a result of softener regeneration, which produces effluent water with exteremely high mineral concentration (extremely high water hardness). This effluent finds its way into water treatments facilities which face challenges in removing the dissolved minerals.

The regeneration process is triggered in the water softeners based on a threshold on the aggregate water flow through the softener since the last regeneration. This is referred to as the "reserve capacity algorithm". Since the regeneration process is blind to the actual hardness in the water, there are three possible cases:

1. *Case 1:* The regeneration happens **after** the softener has depleted its capacity, allowing hard water to reach the consumers and appliances. This is undesirable.

2. *Case 2:* The regeneration happens **before** the softener has depleted its capacity, resulting in higher regeneration frequency and increased softener effluent discharge to water treatment facilities. This is undesirable.

3. *Case 3:* The regeneration happens at the **exact** time when the softener is close to depletion, achieving balance between frequent regenerations and hard water mitigation.

Case 3 does not occur in commercial water softeners since the softeners are programmed pessimistically and favor frequent regenerations (Case 2) to minimize the risk of hard water reaching the consumer (Case 1). As a solution to this pessimism, the AWESOME system [20] provides a mechanism to control the tradeoff between Case 1 and Case 2. The AWESOME [20] system is an example of an Edge Computing platform where the computation and intelligence is decentralized by leveraging compute power at the "edge" of connectivity (e.g. using WiFi routers in the home). In the case of AWESOME, Machine Learning algorithms are deployed at the edge, using edge computing resources. There are two key Machine Learning challenges involved in the solution.

Water Hardness Detection: The hardness is measured using Calcium ion (Ca^{2+}) sensors at the output of the water softener. However, the sensors do not measure a clean, trivially recognizable hardness signal. The measurements exhibit several "false" hardness events, where the signal shares characteristics with a true hardness event. We built a Support Vector Machine to distinguish between the false hardness events and the true hardness events. When a true hardness event occurs, it is an indication of the depleted state of the water softener.

Water Flow and Hardness Forecasting: The hardness is highly dependent on the water flow patterns of the building. Once it is possible to

distinguish between true and false hardness events from the sensor signals, predicting the flow patterns of a building would allow to forecast the exact time of the hardness event ahead of time. This allows flexibility in better protecting the consumers and appliances (Case 1) while minimizing effluent discharge (Case 2).

2.2 Water Hardness Detection

The objective is to devise features and learn a Support Vector Machine (SVM) that is able to distinguish between true hardness events and false hardness events in the water at the output of the water softener.

Feature Engineering

The following features were chosen by empirical studies as being most "informative" towards this classification task.

1. *Measured Hardness:* This is directly calculated from the output of the Calcium ion (Ca^{2+}). The sensor output is a voltage value which is converted to the corresponding hardness measurement in grains per gallon [5].

2. *Flow Rate:* Flow rate is the average rate of flow over a 2 minute interval (can be considered as instantaneous rate of flow considering that the dynamics of the system are extremely slow). In this case, the "instantaneous" flow has been chosen as a feature in order to make the algorithm independent of the cumulative flow, unlike the reserve capacity algorithm. The reserve capacity algorithm uses the cumulative flow or the number of days since last regeneration to trigger the next regeneration [17].

3. *Temperature:* This is chosen as a feature because the output voltage of the Calcium ion (Ca^{2+}) sensor was found to fluctuate with the ambient temperature.

4. *Derivative of the voltage output of the Calcium Ion (Ca^{2+}) sensor:* This is used to encode some characteristics of the shape of the hardness measurement transient.

5. *Temperature compensated voltage output of the Calcium Ion (Ca^{2+}) sensor:* The relationship between the temperature and the voltage output of the sensor is highly nonliner and difficult to characterize. This hand engineered feature was included as a simple compensation technique for temperature effects to help the SVM improve its classification accuracy. This approach is similar to using kernel functions.

The features are also described in 2.1. An example trace of raw hardness data is shown in Figure 2.2.

Data Collection and Learning

The data was collected from multiple installations of the AWESOME [20] system over several months (with at least 2 instances of true hardness events from each installation). [1]

SVM is a supervised learning algorithms, meaning that it requires a training phase in which input data labeled with ground truth is fed to the algorithm. When constructing the training sets for SVM, each feature vector with ground truth. I developed a web-based training tool for time-series data that allows for crowdsourced labeling. A screenshot of this tool is shown in in Figure 2.1. This training tool was used to directly generate data sets for SVM training.

[1] The training was done after the appropriate pre-preprocessing using the `scikit-learn` Python library.

Feature	Formula	Units
ISE Output	$[Ca^{2+}] = e^{k(V-a)}$	$[grains/gallon]$
Flow Rate	f	$[gal/minute]$
Temperature	T	$[^{\circ}C]$
Derivative of ISE Output	$\frac{d[Ca^{2+}]}{dt}$	$[grains/gallon/minute]$
Normalized ISE Output	$[Ca^{2+}]\left(1 - e^{-kf/T}\right)$	$[grains/gallon]$

Table 2.1: Description of hand-engineered features used to detect hard water

This labeling process is subjective because the water hardness — concentration of Calcium and Magnesium ions — is a real number which must be converted to a binary value (0 or 1). So to label training data, a **training threshold** must be applied, below which water is considered to be soft (0), and above which it is considered to be hard (1). The threshold applied will affect the SVM's ability to distinguish between soft and hard water. If the training threshold is too low, the algorithm may classify all unknown inputs as hard (false positives). If the training threshold is too high, all hardness events may be ignored (false negatives). The objective is to produce training sets that will identify filtration medium depletion early (before the outgoing water has become too hard) while rejecting noise. This is discussed further in Section 2.2.

Evaluation

1. How quickly can the SVM identify depletion of the water softener (true hardness event)?

2. How resistant is each algorithm to anomalous sensor inputs (noise)?

The answer to the first question will give insight that is similar to a false-negative rate. An important figure of merit is the amount of time it takes the system to detect water softener resin depletion. The earlier the

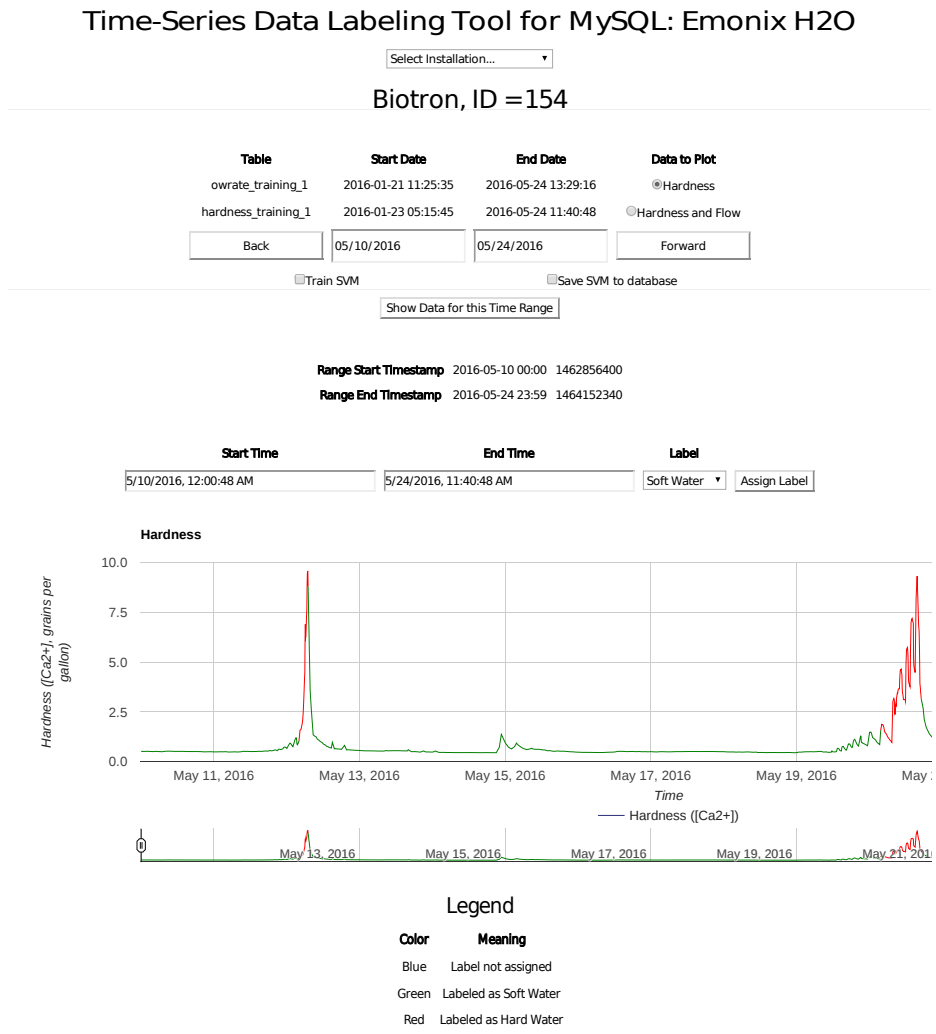


Figure 2.1: Screenshot of Time-series data labeling tool developed for Water Hardness Labeling

detection happens, the lesser the impact of hard water on the residence or industrial facility. The longer it takes the SVM to detect that the resin has been depleted, the harder the water that is produced by the softener, potentially endangering building systems.

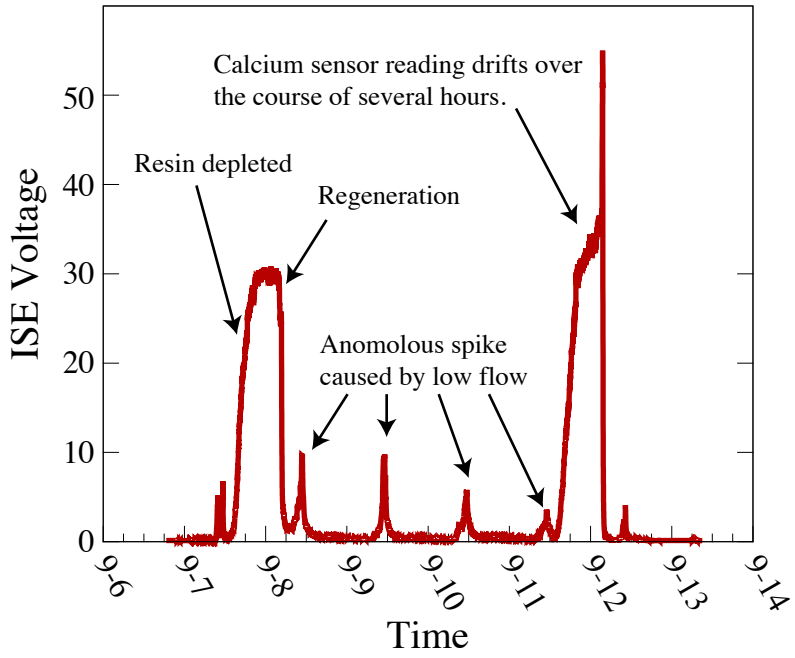


Figure 2.2: Example trace of raw water hardness data showing both true and false hardness events

The answer to the second question will tell us the false positive rate. When the SVM produces a false positive, it regenerates the softener unnecessarily, wasting salt.

To identify filtration medium depletion, it will be necessary to allow the medium to deplete and start producing hard water. The goal is to minimize the amount of hard water that needs to be produced before the SVM can identify that the medium is depleted. If too much hard water is supplied to a building before initiating a regeneration, the water heater and other building systems could suffer damage.

In order to make the SVM detect hard water as early as possible, the training threshold (described in Section 2.2) is manually adjusted based on empirical performance feedback. The lower the training threshold is set, the earlier the SVM will be able to detect filtration medium depletion.

Figure 2.3 shows a plot of the average outgoing water hardness level at which the algorithm can detect medium depletion for several different training thresholds. The figure presentes a comparison of the hardness detection level between simple thresholding and the SVM (simple thresholding has a high false positive rate, as evident from Figure 2.2). The y-axis gives the average water hardness required for each algorithm to identify a filtration medium depletion. To be practical, the detection level should be below five grains per gallon because allowing the hardness to get any higher would damage building systems.

At low training thresholds, the learning algorithms tend to produce more false positives. Figure 2.4 shows the false positive rates on the y-axis with the algorithm on the x-axis. To strike a balance between minimizing false positives and minimizing the hardness detection level, it is best to train the learning algorithms to identify depletion at three grains hardness.

The receiver operating curve (ROC) shown in Figure 2.5 shows the tradeoff between false positives and true positives for SVM and thresholding. On the y-axis, true positives are plotted as a function of false positives. The SVM performs much better than thresholding because we only need to increase the proportion of false positives to 30% in order to achieve 90% true positive detection.

2.3 Water Flow and Hardness Forecasting

The objective is to forecast water flow and water hardness at the output of the water softener based on historical patterns. Forecasting the water flow based on historical patterns is equivalent to modeling the water usage behaviour of a building (residential or industrial). Forecasting the water hardness based on historical patterns is equivalent to modeling the physics of the water softener itself and its depletion characteristics.

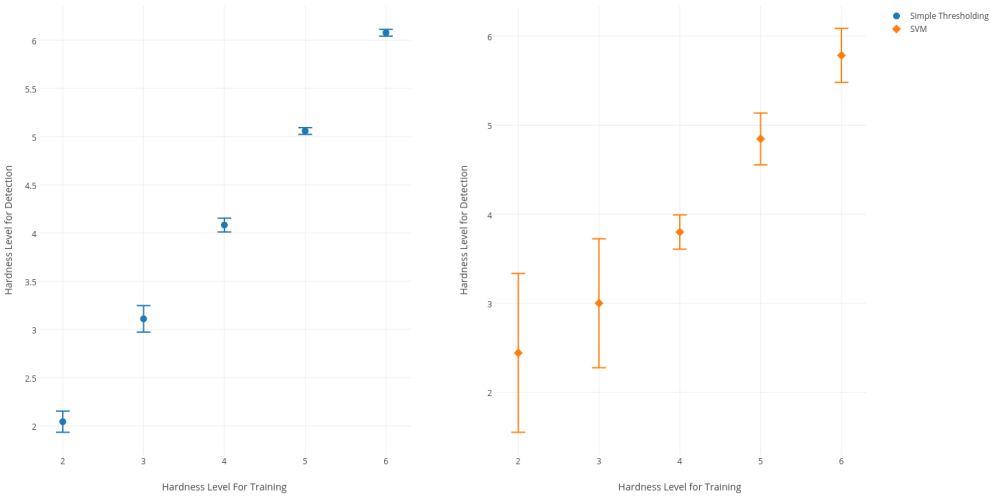


Figure 2.3: The water hardness level of detection for five different training levels.

Water Flow Forecasting

A statistical analysis of water flow patterns in various buildings indicate that the flowrate function is not a stationary process over long time periods. [2] This finding makes sense because activities in a building will likely follow daily circadian fluctuations. At 4 AM, when most residents are asleep, the histogram indicates that the recorded flow rates are mostly zero. Flow rates increase around 8 AM when residents wake up to go to class. At noon, flowrates increase further because lunch is served in the building's cafeteria. Since the distribution of water flow rates changes during the course of the day, it is reasonable to conclude that the process is nonstationary.

Since the flowrate signal is nonstationary, the modeling approach should deal with data that has time-varying distribution. An autoregres-

[2] A stationary process is one whose statistical properties do not change as a function of time. In other words, X_t is stationary iff $F_X(x_{t_1}...x_{t_k}) = F_X(x_{t_{1+l}}...x_{t_{k+l}})$

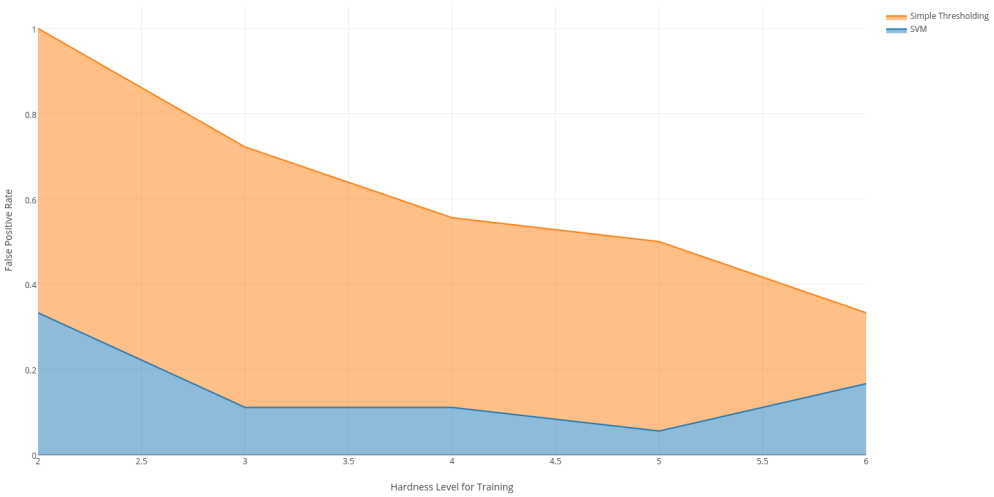


Figure 2.4: False positive rates for the water hardness detection learning algorithm, tested with five different training thresholds.

sive model is a commonly used tool to forecast stationary processes [26]. Here, the autoregressive approach is adapted to work with a nonstationary process, leveraging continuous data measurements.

The flowrate signal tends to be bursty — it tends to be high for a short time when a fixture is turned on and low when the fixture is off. In large buildings, that burstiness is contributed by hundreds of fixtures, each contributing to a noisy signal with a lot of high-frequency components. To deal with such a large magnitude of high frequency components, an extremely high order of autoregressive model will be required, which makes the modeling infeasible. Instead of forecasting the raw flowrate, forecasting the cumulative flow is a reasonable compromise (equivalent to passing the flowrate signal through an integrator). The resulting autoregressive model will have poor accuracy in forecasting instantaneous flowrate. However, this cumulative flow signal is a more useful metric anyway — the cumulative flow is the more important metric for understanding softener

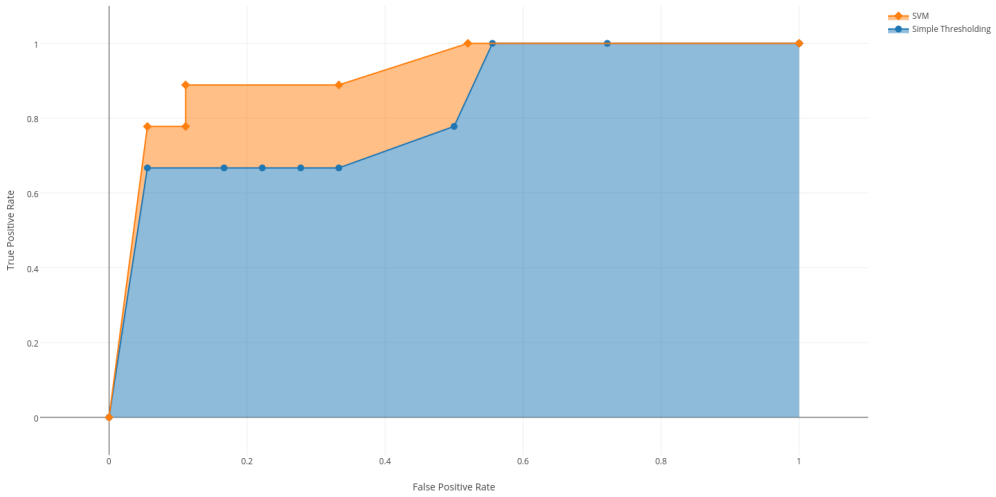


Figure 2.5: Receiver operating curve for the water hardness detection learning algorithm.

depletion.

The autoregressive model then predicts the cumulative flow used by the building on a minute-by-minute basis, 24 hours into the future. The results of the predictions made by the autoregressive model will be presented in Section 2.3.

Water Hardness Forecasting

In addition to forecasting flow, it is also beneficial to predict water hardness readings, since it will be the better indicator of softener depletion. If the water softener's filtration medium will not be depleted in the near future according to predictions (i.e. water will not be hard), there is no point in regenerating the water softener.

Similar to the flow forecasting technique, a vector-state autoregressive model is used to predict future cumulative flow and hardness together. Unlike the flow rate signal, the hardness signal is sparse. Most of the

hardness readings are zero or nearly zero—this is expected because when the water softener is working properly, it removes all hardness from the water. It is only when the water softener's filtration medium depletes that nonzero hardness is observed. The sparse nature of the hardness readings and the inherent nonlinearity of the hardness with respect to the cumulative water flow through the softener poses a challenge for learning linear, stationary models, such as, autoregressive models.

To account for the inherent nonlinearity so described, two operating point models are used (for low/high cumulative flow since the last regeneration). This is justified on the basis that the nonlinearity of the hardness measurement only manifests itself in the high flow region. A parallel can be drawn between this approach and the general practice of linearizing nonlinear systems near typical operating points for the design of closed-loop controllers. Since the hardness measurements depend on the cumulative flow through the softener since the last regeneration, in addition to the history of the hardness measurements themselves, the model can be represented in the following discrete-time state space notation ($\mathrm{flow}(\dots)$ refers to the cumulative flow since the last regeneration):

$$X(k+1) = \begin{bmatrix} A_{\text{flow-flow}} & 0_{\text{flow-hardness}} \\ C_{\text{hardness-flow}} & D_{\text{hardness-hardness}} \end{bmatrix} X(k) \tag{2.1}$$

$$X(k) = \begin{bmatrix} \mathrm{flow}(k) \\ \dots \\ \mathrm{flow}(k-20) \\ \mathrm{hardness}(k) \\ \dots \\ \mathrm{hardness}(k-20) \end{bmatrix} \tag{2.2}$$

$A_{\text{flow-flow}}$ captures the nature of the water usage by the residents of the building. Since this quantification is subject to rapid, unpredictable

changes (nonstationarity), the forecasting model is learnt online after each regeneration cycle thereby using the most recent water usage patterns for the forecasting, eliminating bias due to historical data and addressing the nonstationary nature (resets due to regenerations).

The dependence of hardness measurements on their own history and the flow is quantified by the coefficients matrices of the mode $C_{\text{hardness-flow}}$ and $D_{\text{hardness-hardness}}$. The model so described is learned from recorded data for two operating regions (low/high cumulative flow since the last regeneration). When the actual forecasting is done, one of the models is chosen based on the flow since the last regeneration at the time of forecasting.

Evaluating Forecasting Performance

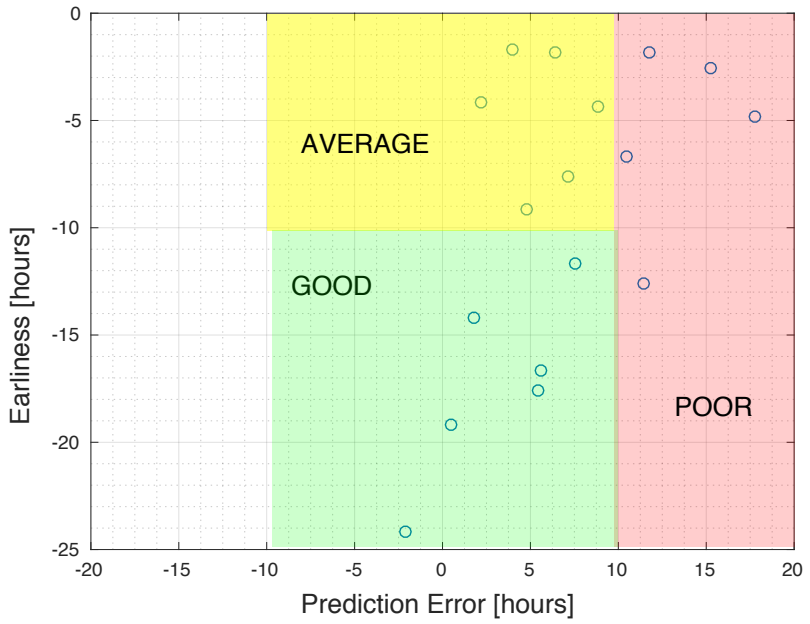


Figure 2.6: Prediction error (X-axis) vs Forecast Horizon (Y-axis) for the hardness forecasting algorithm

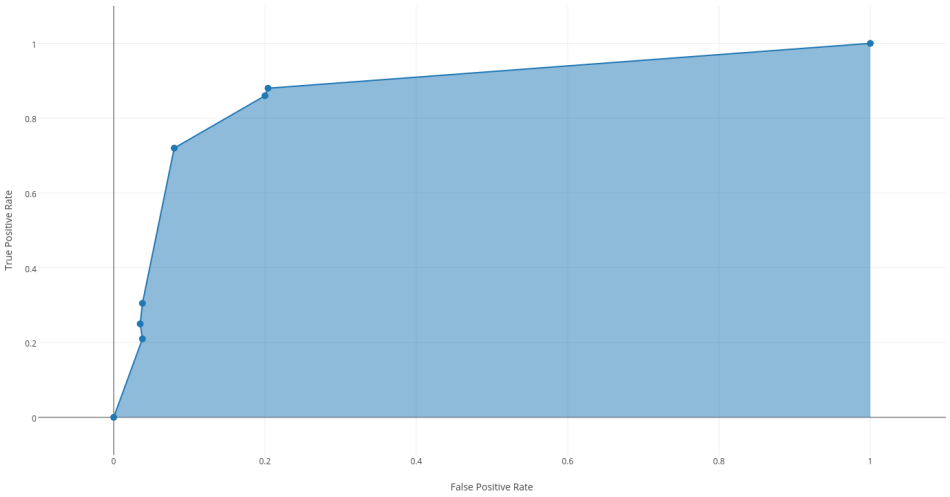


Figure 2.7: Receiver operating characteristic for water hardness forecasting.

Relative prediction error is defined as

$$e_{p_r}(h) = \left| \frac{p(t-h) - s(t-h)}{s(t-h)} \right|$$

where h is the number of hours in advance the algorithm is predicting, $p(t)$ is the prediction as a function of time, and $s(t)$ is the actual measured sensor value as a function of time.

Relative prediction error for cumulative flow is on average below 30% and decreases for longer time horizons. The instantaneous flow signal tends to be very bursty—as fixtures turn on and turn off in the building, water flow starts and stops sporadically. If these bursts do not arrive at the exact moments that the algorithm expects them to, but instead arrive several minutes before or after, then the near-term cumulative flow error will be higher. As long as the bursts of flow eventually arrive, the long-term cumulative flow is low because it captures all the flow that has happened

over a long period of time, and the errors in the prediction cancel each other out.

The hardness forecasting algorithm is intended to predict whether or not water samples will read hard in the next 24-hour time horizon. Since our water hardness sensor readings are sparse—long stretches of zero readings followed by short periods of hard water readings—the objective is to predict not only the exact value of the hardness sensor readings but also the presence or absence of hard water. If, during a forecasting horizon, the hardness value is not predicted accuractely but the presence or absence of a hard water event is correctly identified, then the algorithm has largely achieved its goal.

A plot of the prediction error vs the forecast horizon is shown in 2.6. On average, the relative prediction errors are close to zero for time horizons up to 24-hours in the future. The variance, however, increases for predictions made further into the future.

A receiver operating curve for the hardness forecasting algorithm is shown in Figure 2.7. The ROC was generated by varying the threshold at which water is classified as being hard—higher thresholds resulted in a higher false positive rate.

2.4 Conclusion

In this chapter, I describe techniques to classify water hardness based on ion concentration sensors to determine the state of water softener depletion. In addition, I also describe techniques to adapt commonly used autoregressive models to nonstationary, nonlinear processes with a known nonlinearity (hard resets, sudden water hardness events). The adaptation is required to keep the order of the autoregressive model low, which in turn keeps learning and forecasting computationally light.

3 WIRELESS SPECTRUM ESTIMATION USING SPARSE MEASUREMENTS

3.1 Introduction

Existing spectrum databases [6] [10] [7] in the USA are populated based on RF propagation models. Consequently, the databases are likely to have inaccurate estimates of spectral power and occupancy. There have been wealth of spectrum measurement efforts aimed at recording ground truth data of spectrum usage to better quantify utilization and occupancy and improve the propagation models. These efforts have seen a lot of interest in the context of cognitive radio applications and dynamic spectrum management which are approaches to maximize spectrum utilization [30] [22] [34] [19].

An important concern in deploying devices that support dynamic spectrum access is the identification of areas where spectrum utilization experiences a lot of temporal variations. Applications supporting dynamic spectrum access in such areas can take advantage of temporal variations in spectrum utilization to improve their connectivity. For example, small cell deployments supporting dynamic spectrum access can take advantage of different under-utilized frequency bands during the course of a day and improve backhaul bandwidth. Clearly, deployments supporting dynamic spectrum access call for online spectrum measurement for feedback to minimize the chances of interfering with other applications.

The first step to realize such deployments is to identify areas that see significant temporal variations and pick the best candidates from this information. I propose a method to study temporal variations in spectrum utilization by uncovering missing samples from opportunistic spectrum measurements. I show that opportunistic spectrum measurements are sparse by nature and propose three sources of sparsity associated with

these measurements. Prior work by Zhang and Banerjee [38] [39] discusses opportunistic spectrum measurements as a way to measure 'anchor points' to improve the information quality in spectrum databases. They describe a system 'VScope' which uses a spectrum analyzer placed in a public transit vehicle, to collect spectrum measurements across a city. I reuse the data gathered from VScope [1] and examine its characteristics and propose solutions that uncover the missing samples with high accuracy. I also note that, with large amounts of spectrum data that has been gathered across the world [28] [22] [34] [19] [6] [7] [21] [36] I believe this approach would be useful to study temporal variations in spectrum utilization.

In this chapter, I discuss applying a technique called Matrix Completion [12] (from Collaborative Filtering literature) to uncover the missing samples. I justify the applicability of this technique and evaluate its performance on 6 months of sparse spectrum measurements collected in the City of Madison. I show that even with simplifying linearity assumptions on the spectrum measurement data, it is possible to uncover missing samples with an error less than 5 dBm in about 80% of measurements.

By uncovering missing spectrum measurements using this approach, studying the dynamics of spectrum usage and trends in mobility is made possible. For wireless infrastructure development surveys it is useful to quantify the impact of new infrastructure based on location. I see value for this approach in this context as well, since uncovering missing samples allows for inferences based on a complete data set. I also see value in regulatory efforts for wireless spectrum since this approach could potentially uncover violations in spectrum usage. Additionally, this approach has low computational overhead and recent advances in Machine Learning allow for distributed implementations [29] [27].

Sparsely sampled crowdsourced measurements reduce the burden of deploying comprehensive spectrum measurement devices. However,

[1] I refer to the data gathered as part of the VScope [38] project as "VScope data" for the rest of the thesis.

because of the missing samples, crowdsourced data or opportunistic measurements cannot be used "as is" for analyses. Our approach helps uncover actionable data from crowdsourced spectrum measurements [40] with a computationally light algorithm that can be implemented on mobile devices.

3.2 Challenges in Large-scale Spectrum Measurements

Power measurements for wireless spectrum in the Cellular and Wi-Fi frequency bands are highly dependent on the concentration of mobile devices (smartphones, tablets etc.). Consequently, these measurements exhibit strong temporal and spatial variation, governed by the mobility of users, the variety in types of devices and associated service providers. Studying the types of devices, most devices support multiple cellular frequencies and almost always support Wi-Fi at 2.4 GHz (newer devices also support Wi-Fi at 5 GHz). The support for multiple cellular frequencies in mobile devices is driven largely by the incompatible frequency bands used by the service providers. The Cellular Telecommunications and Internet Association (CTIA) [4] [35] lists more than 20 facility-backed wireless service providers who operate their services in different frequency bands with only few bands overlapping across multiple providers.

What cellular frequencies should we choose to study?

Table 3.1 shows the cellular frequency band allocations for the top five wireless providers in the US who collectively account for over 400 million subscribers. There are more than 50 virtual operators which use the networks of these facility-backed wireless providers to provide service [35]. As a result, it seems reasonable to confine this analysis of the cellular

Table 3.1: Cellular Frequency Band Allocation to the Top 5 Wireless Service Providers in the USA. [35]

Carrier	2G Frequency in MHz Band name			3G Frequency in MHz Band name		4G LTE Frequency in MHz Band number								
	800	850	1900	850	1900	L700	L700	U700	800	850	1700 2100	1900	2300	2500
	SMR	CLR	PCS	CLR	PCS	12,17	29	13	26	5	4,66	2,25	30	41
AT&T	No	GSM	GSM	UMTS	UMTS	Yes	No	No	No	No	Yes	Yes	Yes	No
T-Mobile	No	No	GSM	No	UMTS	Yes	No	No	No	No	Yes	Yes	No	No
Sprint	CdmaOne	No	CdmaOne	No	CDMA2000	No	No	No	Yes	No	No	Yes	No	Yes
Verizon	No	CdmaOne	CdmaOne	CDMA2000	CDMA2000	No	No	Yes	No	No	Yes	Yes	No	No
U.S. Cellular	No	CdmaOne	CdmaOne	CDMA2000	CDMA2000	Yes	No	No	No	Yes	Yes	Yes	No	No

spectrum to the frequency bands from Table 3.1 as representative of the cellular spectrum usage.

What are the challenges in city-wide measurement of wireless spectrum?

For modeling these power measurements, RF propagation models are typically used to generate a power estimate. However, this technique lacks accuracy in city settings since buildings and multipath effects introduce complications in the model. Using detailed 3D terrain and building data, some authors have proposed to improve the modeling estimates [18] [16]. However, such approaches requires significant computational resources and modeling effort for generating power estimates. Additionally, some frequency bands show significant temporal variation in power as described in [19] rising from the dynamics of users carrying mobile devices. Using RF propagation models would not capture such local, temporal variations of the spectral power inside a city, since the methods rely on the power output from the transmit antenna and ray-tracing simulations.

My goal here is to discuss a method to study temporal variations in spectrum usage that uses, comparitively, much lesser computational resources while simulatenously providing more accurate measurements

than propagation models.

Can opportunistic spectrum measurements help understand temporal variations?

To address inaccuracy issues with RF propagation models, Zhang and Banerjee [37] [38] [39] propose VScope, an opportunistic spectrum sampling method to obtain ground truth power measurements, which can serve as "anchor" points to improve modeling accuracy for TV whitespace spectrum. VScope involves collecting measurements from a spectrum analyzer placed in a moving public transit vehicle. This method has the advantage of not requiring complicated terrain or building models to generate the power estimates. Instead, VScope chooses to use empirical data to correct estimates from RF propagation models. By augmenting such "ground truth" measurements combined with diversity in measurement locations (as a result of the moving vehicle on which the spectrum analyzer is mounted), VScope is able to improve the accuracy of the power estimates with a less computationally intensive approach. My work builds on top of the results from VScope and uncovers temporal variations from sparse measurements.

The VScope data has been collected for a 6 month period from July 2015 - January 2016 and has samples covering the entire city of Madison. Data has been collected for the frequency range 300 MHz - 3 GHz with a precision of 4 kHz in the frequency domain. It takes the spectrum analyzer approximately 7 minutes to do a full sweep of measurements from 300 MHz - 3 GHz (the complete frequency range for measurement). These measurements include the frequencies of interest as identified in Section 3.2.

I define sparsity of measurements here as less than 5 power measurements for a given frequency band within a given hour of the day at any location within a city. The opportunistic spectrum measurements from

VScope are sparse by nature; the sparsity is induced by the varying routes, schedules and motion of the public transit vehicle. It is possible to identify three types of sparsity in the VScope data, as listed below.

Motion Induced Sparsity Because of the motion of the bus and the 7 minute interval it takes the spectrum analyzer to sweep the entire frequency range of measurement, a given location has very few samples within any one hour period. Additionally, the samples so collected only correspond to a small portion of the complete frequency range (the measurements recorded while the transit vehicle is in the "vicinity" of the location).

Schedule Induced Sparsity The vehicle carrying the spectrum analyzer is typically assigned to a single route for a specific day. However, the times during which the vehicle actually passes through a given location depends on both traffic conditions and schedules assigned by the transit authority. Consequently, a given location has measurements recorded only at certain hours within a single day.

Route Induced Sparsity The vehicle carrying the spectrum analyzer is regularly reassigned to different transit routes within the city. Consequently, when it's required to analyze weekly, monthly or seasonal trends at a single location, the number of measurements is low in number across different days.

Owing to this sparsity, the data provides us partial information about temporal variation of spectral power, at a given location. Because of the missing samples, the data cannot be used "as is" to study temporal variations.

How do we address the sparsity in spectrum measurement?

Key Intuition: Given that only cellular and Wi-Fi frequencies are being studied, my conjecture is that the samples collectively show strong linear relationships. This intuition stems from two factors:

- The mobile devices that cause local variations in the spectral power are typically active in multiple frequency bands at the same time. For example, smartphones (that form a large share of common mobile devices) are typically active in both the 4G LTE bands as well as the Wi-Fi bands.
- The circadian rhythms of people moving through a particular location causes repetitions or simple linear patterns in temporal variations. For example, a large majority of people work 9 to 5 jobs and are likely to cross the same location twice a day (repetition).

Therefore, we can use this simplying assumption of linear relationships between spectrum measurements to estimate the missing samples from the observed samples.

Can we collect better data? Or is the sparsity unavoidable?

The motion induced sparsity is especially interesting because it is caused, in large part, owing to the 7 minute interval the spectrum analyzer takes to do one full sweep of the frequency range. Therefore, an improvement in the measurement cycle with better instrumentation can significantly decrease the sparsity induced resulting from the motion of the public transit vehicle, which in turn would help build better temporal models.

In contrast, the sparsity in measurements induced by the varying routes and schedules of the public transit vehicle are uncontrollable factors and have to be treated as an inherent sparsity in the measurements. This stems from the fact that we deployed only one spectrum analyzer in a single

public transit vehicle. If we had multiple spectrum analyzers deployed in multiple vehicles, this sparsity would be reduced. However, deploying multiple spectrum analyzers comes with the overhead of increased cost since such high resolution spectrum analyzers are quite expensive. Consequently, the Schedule and Route induced sparsities can be attributed to a trade-off between deployment cost and quality of data.

A completely different angle to solve this would be to employ solutions such as WiSee [40] which bring the capabilities of a spectrum analyzer to the smartphone. Such an approach would allow for crowdsourced acquisition of spectrum measurements which could potentially provide data with much lower sparsity, depending on the adoption of the approach. Even in this scenario, we believe that the methodology discussed in this chapter could prove valuable in estimating temporal variation in spectrum measurements.

3.3 Applying Matrix Completion to Estimate Missing Samples

In this section, I propose a method to address the Route Induced Sparsity in spectrum data measurement.

Assumptions

To make the analysis tractable, I make the following assumptions.

1. Spectrum measurements remain the same in a 200 m x 200 m square area

2. There is an underlying linear relationship between the sampled data points across time and frequency, at a single location

3. The power measurement remains constant in a one hour time window. In other words, a single measurement during any hour of the day is considered to be representative of the power measurement for the entire hour.

Assumption 1 is justified because most cellular frequencies possess this range, considering that all the VScope measurements are concentrated on the road within a small area (no decay owing to buildings or terrain).

Assumption 2 is justified based on my conjecture in Section 3.2.

Assumption 3 is not a great assumption, since it ignores the spectrum dynamics within a single hour (from our definition, at least 5 measurements within the hour qualify as sufficiently dense). However, since the objective is to address the Route Induced Sparsity, there is some merit to this simplification. Since the spectrum analyzer is placed on a public transit vehicle, the samples are likely to be collected at times when there is a significant amount of user mobility i.e. the samples are likely to be at the "interesting" points of time for the dynamics of spectrum usage. Moreover, I demonstrate good performance of this technique even in this case, which possibly indicates that the temporal dynamics of the spectrum within the hour may not be too significant.

Implementation

I organize the VScope data in the following manner:

- We arrange the data in a matrix form, with each row representing the hour of day and each column representing 10 MHz frequency bands.
- Since the highest concentration of user traffic and public transit occurs between 8 AM and 6 PM, I confine the analysis to this time

window within a certain day. This is also the same window in which most of the VScope samples are concentrated in.

- The spectrum analyzer measures the power in 100 MHz chunks with a precision of 4 kHz (26215 values). We reduce this to 10 values (corresponding to 10 MHz each). The reduction is done by computing the mean power in the 10 MHz band
- The frequencies that I choose to form are matrix are 100 MHz bands starting at the following frequencies: 300 MHz, 500 MHz, 600 MHz, 700 MHz, 800 MHz, 1800 MHz, 1900 MHz, 2100 MHz, 2400 MHz, 2500 MHz
- The resulting matrix is of size (10, 100)
- We include the 500 MHz and 600 MHz TV spectrum bands since they are fairly constant and help the algorithm "connect" or infer relationships between multiple rows with small number of measurements

The matrix so constructed should be of low rank owing to the linear relationship between the samples (Section 3.2). As a result of this, it is possible to use the low-rank matrix completion technique proposed by Candes and Recht [12] to uncover the missing samples. At the core of this approach is the expectation that the matrix is low-rank, which stems from my conjecture about linear relationships between samples. I implement the technique using the algorithm proposed in [11].

3.4 Evaluation of Matrix Completion on Sparse Opportunistic Measurements

Since the VScope data does not provide all entries in the matrix, I choose to evaluate this technique as follows:

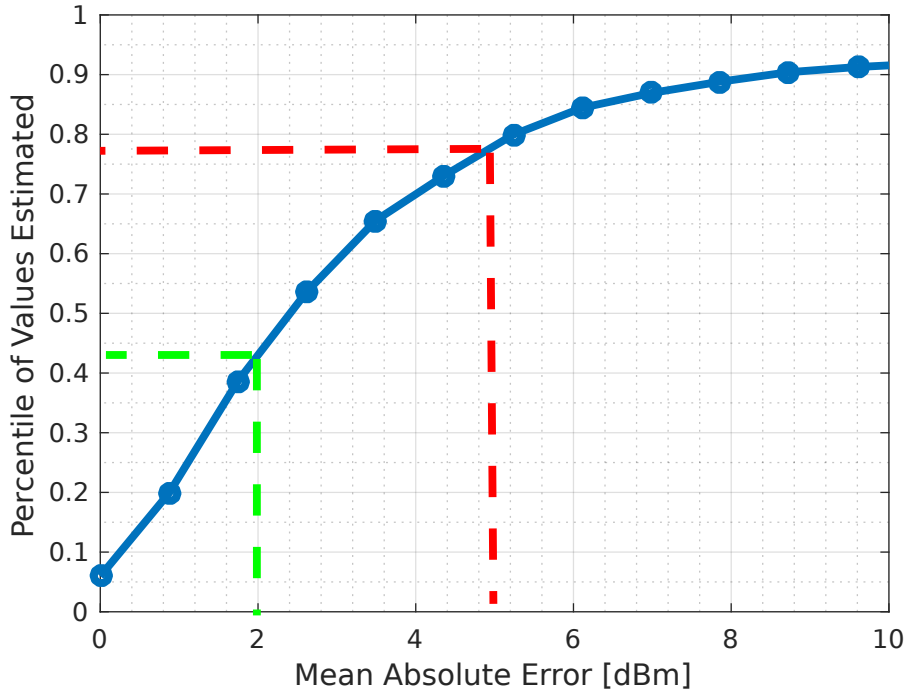


Figure 3.1: Percentile of estimated wireless spectral power values against Mean Average Error of the estimates

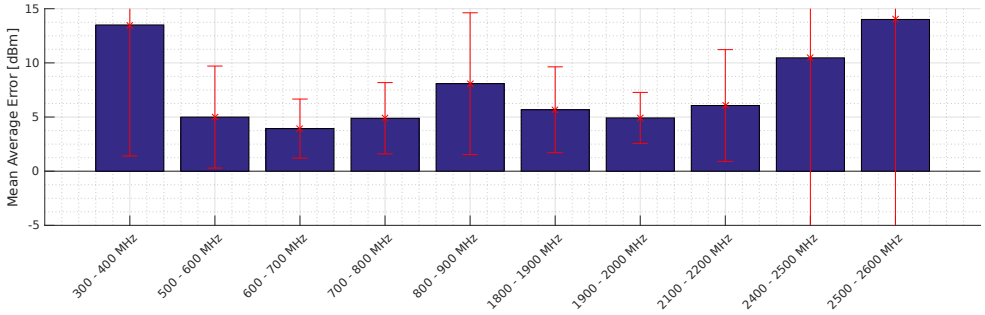


Figure 3.2: Mean Average Error of the estimated wireless spectral power for each 100 MHz frequency band

- Construct the matrix for a single day, at a single location
- Pick a 100 MHz frequnecy band for which measurement exists and

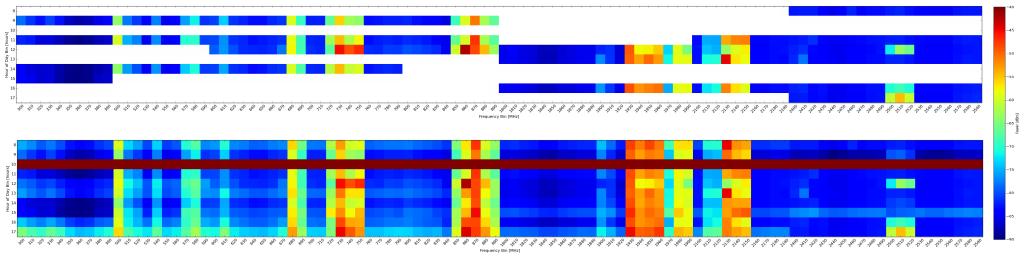


Figure 3.3: Image Comparison of incomplete matrix (ground truth samples only) and uncovered matrix. X-Axis: Each entry along a row corresponds to the total power in a 10 MHz frequency band. Y-Axis: Each entry along a column corresponds to an hour of the day betwee 8 AM and 6 PM. Color: Each shade of color corresponds to a different power value as defined by the colorbar on the right

hold-out the corresponding 10 samples

- Run matrix completion on the rest of the samples and compare estimates with the 10 samples that were held out
- Repeat this process for every 100 MHz frequency in the matrix and collect results about the estimation accuracy

I used this evaluation procedure for 5 different locations in Madison that observe both high and low foot traffic. For each location, I evaluate the algorithm for 4 different days (a total of 20 matrices and 20,000 entries). Since the chosen locations capture diversity in the underlying mobility models, I believe that this evaluation is representative of the general performance of this technique.

Figure 3.1 shows the percentile of data points against the mean absolute estimation error in dBm. This is useful to understand the prediction accuracy of the algorithm across frequency bands and builds confidence in the low-rank structure conjecture. The estimation error is < 2 dBm at the 40th percentile and < 5 dBm at the 80th percentile of the measurements.

Figure 3.2 breaks up the error metrics by the frequency bins being studied. The mean absolute estimation error is < 5 dBm across most frequency bands.

Of course, these metrics are not fair if all measurements only indicate low spectral power (low occupancy) - if that was the case, then the problem is trivial and would not require such analyses. However, since I am considering cellular and Wi-Fi frequencies in a city setting low occupancy is almost never the case, depending on the exact band being examined. In Figure 3.3 I present a comparison between the measured sample matrix and the completed sample matrix using this method. Each column of the matrix corresponds to a 10 MHz frequency band and each row corresponds to an hour of the day from (8 AM - 6 PM).

We see that the power measurements show variation between -120 dBm to -50 dBm indicating that some bands do have meaningful activity. A glance at the "ground truth" measurements also indicates temporal variations hour-to-hour from which we are able to capture the hidden relationships to predict the missing samples. Additionally, we also witness one of the present drawbacks of this approach : when no samples are present in a one hour window, the algorithm is unable to "connect" the measurements for that hour with the known measurements and fails. This can be observed by the solid horizontal line for the hour between 10 AM and 11 AM.

3.5 Conclusion

I present a simple, computationally light approach to uncover power measurements of wireless spectrum for sparse samples. I evaluate the performance of matrix completion on 6-months of data collected in a mid-size US city. I show that it is possible to achieve good performance even with simplified linearity assumptions which show the promise of this

technique.

This approach can be potentially used to benefit applications such as QuickC [23] which aim at improving wireless backhaul. With large amounts of spectrum measurement data being gathered across the world [30] [22] [34] [19] I believe that this approach would be useful to draw better inferences from spectrum data. Additionally, this approach decreases the measurement burden in acquiring spatially and temporally dense spectrum measurements since I show that it is possble to uncover missing measurements from sparse spectrum samples.

4 PRIVACY-SENSITIVE ACOUSTIC PERCEPTION FOR MOBILE DEVICES

4.1 Introduction

Speech enabled technology is an increasingly popular element in consumer devices, as seen in Smartphones, Home Assistants etc. To provide speech enabled features, devices rely on listening to the ambient raw audio stream for keyword spotting. Applications listening to raw audio pose serious questions about conversational privacy and personal security for the end-user [1]. As an attempt to protect privacy, the speech enabled features have been restricted only to "native" applications written by the device manufacturer.

In this chapter, I present a solution to this problem in two phases. I discuss audio obfuscation techniques and examine the trade-off between privacy and keyword spotting performance. I also identify the best solution within our framework that maximizes both privacy and keyword spotting performance. I present performance results in the field from an Android application (`dBHound Keyphrase`). I also support my argument with human level cognition performance on obfuscated audio. Using this approach allows the entire suite of applications on a consumer device to provide keyword-enabled services while maintaining user privacy. This is achieved by openly broadcasting the obfuscated audio to the applications on the device. Since most user interaction with applications can be decomposed into keywords, I envision that this approach will lead to comprehensive speech-enabled services while maximizing end-user conversational privacy.

4.2 Threat Model and Privacy Objectives

I define the application context as interactions with mobile devices that involve directing speech toward the device. This agrees with typical usage patterns of mobile devices with regards to keyphrase recognition (Figure 4.1). This definition for our application context allows us to leverage difference in the audio signal caused by the directionality of sound waves.

Figure 4.1: Keyphrase Recognition Application Context

I choose a list of common keyphrases 4.1 for evaluating the "utility" (keyphrase spotting performance) of my solution. The ambient noise in the background while using the application can either be white noise or babble noise. I also analyze the effect the intensity of noise has on our keyword spotting performance.

I use the TED-LIUM speech corpus [31] as a representatitve example of everday conversations in English. The corpus also has the advantage of providing English speech examples from speakers with varying accents. I also construct babble noise examples by randomly selecting and merging small audio segments from the TED-LIUM speech corpus, as well as borrowing audio snippets from other noise data sets. For keyphrase data, I have crowdsourced data acqusition through my Android app and have a

okay google	call mom
hey siri	take a picture
hey cortana	play a song
nine one one	make a note

Table 4.1: Common keyphrases chosen for prototyping privacy-preserving algorithms

rich set of examples for each keyword. More details on the data acquisition process is described in Section 4.3.

Rogue Mobile Applications

The user installs an application "A" on his mobile device to assist with a particular task. The application "A" provides a voice interface for everyday use and is granted access to the microphone. However, the application "A", however is malicious and records everyday conversations of the user and transmits the audio content to a remote database. An attacker uncovers sensitive information from the audio in the remote database using speech recognition.

This is a threat model when applications have access to raw audio data.

Adversarial Database Queries

The user installs an application "A" on his mobile device to assist with a particular task. The application "A" provides a voice interface for everyday use and is granted access to obfuscating audio that is sufficient for its voice interface. The obfuscated audio data is stored in a database to improve the voice interface of the application. An attacker has prior knowledge of a keyword whose presence/absence (binary query) would indicate sensitive information of value. The attacker uncovers some sensitive information with a sequence of binary queries using keyword spotting.

This is a threat model when applications have access to obfuscated audio data.

4.3 dbHound: System Design

The dBHound app is designed to simulate a layer between the raw audio stream and voice-enabled mobile applications. The applications only receive the obfuscated audio stream for providing their voice-enabled features. Any application that requires access to raw audio needs to request for permission from the user every time (Figure 4.9).

The dbHound system is composed of an Android application to interface with the user and collect audio data, and a backend server that obfuscates the audio based on the settings and returns the classification result. The app communicates with the backend server using a REST API. The app does not access any personal information and all the REST API calls are completely anonymous. The only personal information collected is what the user provides when volunteering training data for the dbHound classification system.

Figure 4.2 shows the screenshot of the `dbHound Keyphrase` [1] Android application for users to volunteer training data. Figure 4.3 shows the screenshots of the db `dbHound Keyphrase` application for users to test the keyphrase recognition. If the "Classify and Incrementally Train" option is enabled, the trained models are "fine-tuned" based on the user audio and label specified.

For preserving privacy, I propose a two-part solution:

- A computationally light transformation that removes conversational information from audio but preserves the ability to recognize a desired list of keywords.

[1] The app can be found on the Google Play Store at `https://play.google.com/store/apps/details?id=edu.wisc.cs.dbhound`

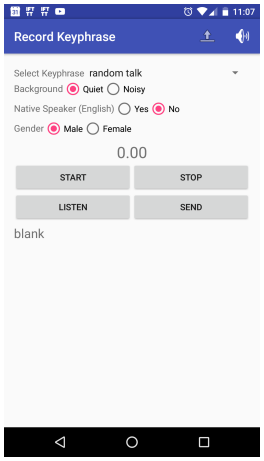
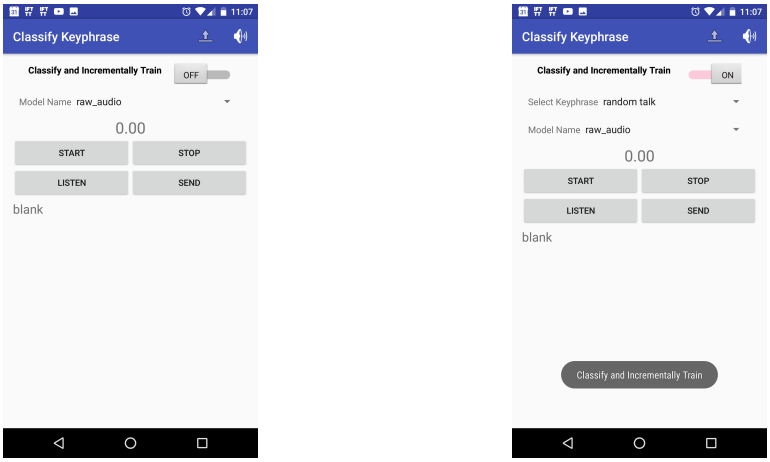


Figure 4.2: `dbHound Keyphrase` Android app screenshot for submitting training data



(a) Only Classify **(b)** Classify and "Fine-Tune" Model

Figure 4.3: `dbHound Keyphrase` Android app screenshot for testing the classification models

- A computationally light inference technique to spot the keywords from a stream of transformed audio data.

4.4 Lightweight Audio Obfuscation

We require a lightweight obfuscation technique that can run in real-time on mobile devices with minimal computational effort. Driven by this constraint, I chose to explore two simple obfuscation techniques: decimation (downsampling/reducing the samplerate of audio) and imprecision (reducing the bit depth of the audio). Both these operations are quick and can run in real-time even in low-power microprocessors [2].

Decimation: Removing time fidelity

If the audio stream is recorded at 16kHz, we decimate the audio stream to a much lower sampling frequency. With this operation, the intelligibility of speech rapidly decreases with the amount of decimation. However, this also significantly decreases keyword spotting performance 4.6.

$$\text{Decimate(Audio)} = \text{Downsample(Filter(Audio))} \tag{4.1}$$

A Weiner filter is used here to process the audio. The Weiner filter works based on the probability of the sample belonging to the signal distribution or noise distribution. Therefore, it requires a specification of the noise power. The noise power is calculated as the variance of the first 100ms of recorded audio. The filter is applied in successive windows of 10ms of audio and the signal power is calculated as the variance of the audio window.

[2] One prevalent example of a low-power microprocessor that is ideal for this application is the Android Sensor Hub [3]

$$P(X[i : i + \text{window\_size}]) \sim \begin{cases} \mathcal{N}(0, \sigma^2_{\text{signal}}), & \text{if signal} \\ \text{or,} & \\ \mathcal{N}(0, \sigma^2_{\text{noise}}), & \text{if noise} \end{cases} \tag{4.2}$$

Bit Depth: Removing amplitude fidelity

If the audio stream is recorded at a 16-bit depth, we quantize the audio stream to a much lower bit depth to reduce the information content. Surprisingly, we find that, even at the lowest possible 1-bit depth, the speech content can be easily understood by humans (Figure 4.4).

Obfuscating audio using hamming weights

If we consider each audio sample as one "sample bit", then we can compute the hamming weight of n sample bits. This is done by concatenating the sample bit strings and counting the total number of "1"s in the n-sample bit string. This operation increases the entropy of the obfuscated audio as compared to the original audio. A simpler way to analyze — for a calculated hamming weight, there are combinatorially many possibilities for the "true" audio sample. The operation is illustrated in the example below.

...	2919	−32006	...

which, in bit representation

...	0000 1011 0110 0111	1000 0010 1111 1010	...

Audio Samples

If we apply hamming reduction on every 1 sample,

...	8	8	...

which, in bit representation

...	0000 0000 0000 1000	0000 0000 0000 1000	...

Hamming Reduced Audio Samples (`hamming_reduce_1`)

As we can see, although both the samples have extremely different values, but their hamming reductions are identical. For a hamming reduced value of 8 the total number of possible "true" sample values with 16-bit audio are $\binom{16}{8=12870}$. Of course, the total number of possible "true" samples values decreases as the hamming value approaches 16. More generally, the amount of privacy improvement resulting from the hamming operation can be defined as follows.

Let the reduced hamming weight value (8 in the example above) be represented by R_s. The number of sample-bits for the hamming operation (1 in the example above) is represented by B_h. The bit-depth of the audio (16 in the example above) is represented by B_s.

$$\text{Privacy Improvement from Hamming Reduction} = \frac{\binom{B_h \times B_s}{R_s} - 1}{2^{(B_h \times B_s)}} \quad (4.3)$$

Higher the order of the hamming operation, higher the privacy improvement. However, the hamming operation does decrease the performance of the classification of the deep learning model, so this is a trade-off that must be examined in detail.

Adding keywords to the dbHound system

Training classification models for new keywords is quite straightforward using the existing dbHound framework 4.3. To add a new set of keywords for the classification task, I propose a cascaded classification topology to avoid re-training older keyphrase models where the older keyphrases may

be phonetically close to the newer keyphrase.

For example, if "hey savannah" is a new keyphrase — since it sounds phonetically close to "hey cortana", a cascaded classification topology can be used. In the cascaded topology, the new keyphrase model is trained to classify all the original keyphrases as "random talk". Consequently, the newly trained model will be evaluated first (in the cascaded structure). If the audio does not belong to any of the new keyphrase model(s), then the audio is evaluated on the older keyphrase models to determine the class label. In a cascaded topology, this evaluation trickles down until the oldest keyphrase models are reached or until a keyphrase label with probability above the confidence threshold is found. This way, the computational expense in model evaluation is also minimized on average.

Human-level performance

Measuring Human level performance on a Deep Learning tasks is often an important metric and potentially leads to interesting insight [8]. Following this paradigm, I selected 52 audio snippets from the TED-LIUM corpus [31] and applied different levels of obfuscation using both the methods described above. Using workers from Amazon Mechanical Turk [2], the ability of humans to understand/decipher the speech content of obfuscated audio tracks was measured. The human transcription from the Mechanical Turk workers is compared against the original transcribed text in the TED-LIUM corpus to measure accuracy. The more accurate the transcription, the less effective the obfuscation technique. Surprisingly, I found that even when the bit-depth of audio is reduced to 1 bit, comprehensibility is quite high in most cases (the exception being audio with low Signal to Noise Ratio [SNR]). The results from the mechanical turk survey is shown in figure 4.4. The first 4 bars correspond to decreasing decimated frequencies. The last four bars correspond to decreasing bit depth of audio.

4.5 Keyphrase Recognition from Obfuscated Audio

I propose a deep neural network architecture (Figure 4.5) to achieve this keyphrase recognition task in real time. Research work from Google in the past [14] [32] [15] [13] have shown that deep convolutional neural networks are suitable for online keyphrase recognition in mobile devices. The focus of my work is to survey the trade-off between privacy and keyphrase recognition performance for different audio obfuscation techniques. Based on the work from Google, I add the remark that efficient implementations of such architectures have been realized for mobile devices and it will be possible to translate the architecture I propose in a similar fashion, to mobile devices to realize the recommendations in Section 4.7.

The different layers of the deep convolutional neural network are described in detail below.

Input Layer

This contains all the samples from the obfuscated audio for a 2.048 second interval, with the audio being sampled at 16 kHz with 16-bit resolution. The interval of 2.048 seconds was chosen so that the corresponding number of raw audio samples is a power of 2 (32768 raw audio samples). This choice is purely one of convenience for signal processing.

Based on the obfuscation technique applied, the total number of samples provided in the input layer may be different. Some examples for this are shown in Table 4.3. However, all of these variants still correspond to a time interval of 2.048 seconds, assisting in the consistency of the deep convolutional neural network across different obfuscation techniques.

For the sake of clarity when working with convolutional layers, the shape of the input layer can be re-written in the following manner.

$$\text{Shape Template} = (\text{height} \times \text{width} \times \text{channels}) \tag{4.4}$$

$$\text{Shape of Input Layer} = (\text{N} \times 1 \times 1) \tag{4.5}$$

Convolutional Layers

In this case, I propose using 3 cascaded convolutional layers (interspersed with 3 max-pooling layers) to learn audio features for the classification task. Since the "width" (from the shape template above) is always 1, these layers can also be likened to a 1-D convolutional operation. Each convolutional layer is completely defined by the parameters described below. [3]

- **Filter Size** (S)**:** The size of the convolutional filter defines the number of samples to work with to generate a feature. This also decides the size of the weight matrix for a single convolutional filter. For example, if only one filter of size $(20 \times 1 \times 1)$ is present, the filter works with 20 audio samples and 1 channel In other words, 20 weight variables are learnt per filter. The filter width is always 1, since this is a 1-D signal, and typically the filter works with all the channels of the input. Consequently, the filter size S can be defined by a single number, 20, in this case.

- **Filter Strides** (P)**:** The term "convolution" applies here because of the striding operation. Once the size of the convolutional filter is decided, the filter walks over sub-sections of the input space to compute a feature for different parts of the input. This can be likened to computing the Fourier Transform of different portions of a signal to understand localized frequency distributions — the convolutional

[3] Chris Olah's explanation in [25] is a great resource to understand the structure of ConvNets.

filter learns to find the best localized features. Clearly, given the high sample frequency of audio signals, striding the filter sample by sample would result in excessive computational time. Moreover, striding sample by sample is not likely to provide accuracy benefits either — owing to the high sample frequency, the information (or phoneme) content is dispersed. The Filter Strides P is the number of samples the filter "walks" over before computing the next feature.

Decreasing this parameter results in the filter striding in shorter steps (more instances of the same filter), increasing computation time while learning and evaluating. The size of the model (in terms of memory), however, remains unchanged.

- **Number of Filters** (K)**:** This is used to set the number of filters (different features) to learn from the input. This can be likened to computing different types of transforms (e.g. fourier, haar, daubechies, cepstral etc.) over the same sub-sections of the input data. The intuition behind this is that different filters are likely to capture the essential features of the signal when used together.

- **Activation Function:** This is the activation function used to generate the output at each node. Typical choices are ReLU, Sigmoid, tanh etc.

The size of the output of the 1-D convolutional filter is described below. The structure of my proposed deep convolutional neural network is described in 4.4.

$$\text{Shape Template} = (\text{height} \times \text{width} \times \text{channels}) \tag{4.6}$$

$$\text{Shape of Input Layer} = (N \times 1 \times C) \tag{4.7}$$

$$\text{Shape of Output Layer} = \left(\frac{N - S + 1}{P}, 1, K\right) \tag{4.8}$$

The max-pooling layers are equivalent to a subsampling operation. They are introduced to maintain local translational invariance in the model. Consequently, a max-pooling operation of 4 decreases the number of input samples by a factor of $\left(\frac{1}{4}\right)$ in the output. The number 4 here is referred to as the "kernel size" of the max-pooling layer. It is standard practice for a max-pooling layer to follow a convolutional layer. [4] All the max-pooling layers used in the architecture in Figure 4.5 have kernel size 2.

Densely Connected Layers

The cascaded convolutional layers are used to learn the "best" features for the classification task at hand. The densely connected layers are intended to learn the actual classification task from the "best" features. If the output of the cascaded convolutional layers are considered as "features", then the rest of the network is a two-layer perceptron. In other words, the rest of the network can be treated as a neural network with one hidden layer and one output layer.

The hidden layer has ReLU activation with dropout regularization (the dropout probability is chosen as 0.8). The output layer has a softmax activation function to compute class probabilities.

When the neural network is fine-tuned online by the dbHound system, only the weights for the densely connected layers (two-layer perceptron) are updated. The reason only the two-layer perceptron is fine-tuned is to prevent overfitting on new data — using this approach, the feature computation is "frozen" and the classification boundary is **slightly** nudged to accommodate new examples. This "incremental" learning is also done with a very small learning rate with the idea that the classifier is already well learnt and only needs to be slightly adapted to deal with new data.

[4] The Convolutional Layer together with its Max-Pooling Layer will be referred to as Conv-Max layer in the rest of this thesis.

4.6 Evaluating Keyphrase Recognition Performance

In this section, I present performance results for the ensemble of classification models learnt from keyphrase data. The model is organized as follows:

- An individiual classification model is learnt for every keyphrase apart from "random talk". The objective of the classification is to distinguish between the true keyphrase and "random talk".
- For a test audio clip, all 8 classification models are used to get class probabilities for the associated keyphrase. The keyphrase with the maximum class probability is chosen and compared to a confidence threshold. If the class probability is greater or equal to than the confidence threshold, the classification output is the true keyphrase. If the class probability is lower than the confidence threshold, the classification output is "random talk".

Summarized in tables 4.5 to 4.18 are the mean class probabilities for different keyphrase audio examples using different keyphrase models. Each table corresponds to a different obfuscation technique, as noted in the caption. Each table corresponds to a different obfuscation technique. Each row corresponds to a classification model for a single keyphrase. Each column corresponds to audio examples for a single keyphrase, or audio examples for "random talk". The class probabilities in the table indicate the level to which the model has learnt the training set.

Examples for "random talk" are an assortment of choices from three sources:

- TED-LIUM Speech Corpus
- Babble Noise Examples (Cafeteria noise, Public Transit Vehicle Noise)

- "random talk" examples from the `dbHound Keyphrase`[5] android application

Figures 4.6 4.7 4.8 show the Receiver Operating Curves (ROC) for different obfuscation techniques. The Receiver Operating Curve shows the variation of the false positive ratio and true positive ratio when the confidence threshold for classification is varied. These curves help understand how well the model is likely to perform in real life.

[5]The `dbHound Keyphrase` android application can be found on the Google Play Store at `https://play.google.com/store/apps/details?id=edu.wisc.cs.dbhound`.

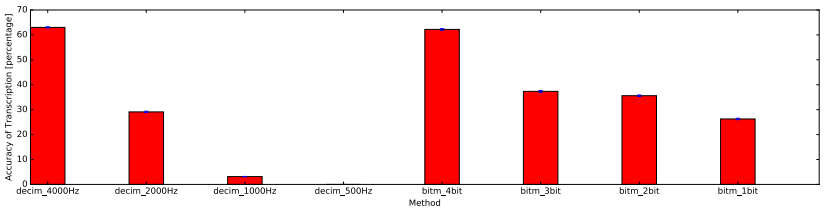


Figure 4.4: Accuracy of human transcription for different levels of decimation and quantization.

Key	Description	Key	Description
decim_4000	Decimated to 4 kHZ	bitm_4bit	4-bit precision audio
decim_2000	Decimated to 2 kHZ	bitm_3bit	3-bit precision audio
decim_1000	Decimated to 1 kHZ	bitm_2bit	2-bit precision audio
decim_500	Decimated to 500 HZ	bitm_1bit	1-bit precision audio

Table 4.2: Key for X-axis in Figure 4.4: Accuracy of human transcription for different levels of decimation and quantization.

Obfuscation Technique	Input Layer Size
No Obfuscation (Raw Audio)	32768
Decimation to 4 kHz	8192
Decimation to 1 kHz with 8-Hamming Reduction	1024
Decimation to 1 kHz	2048
Decimation to 1 kHz with 2-Hamming Reduction	1024
Bit Depth to 1 bit	32768
Bit Depth to 1 bit with 8-Hamming Reduction	4096

Table 4.3: Example list of size of input layer in the Deep Convolutional Neural Network for various obfuscation techniques

Parameter	**Convolutional Layer: 1**	**Convolutional Layer: 2**
Filter Size	$S = \frac{N}{128} \sim 16ms$	$S = 2 \sim 32ms$
Filter Strides	$P = \frac{S}{2} = \frac{N}{256}$	$P = \frac{S}{2} = 1$
Number of Filters	1024	512
Activation Function	ReLU	ReLU
Regularization	L_2	L_2

Parameter	**Convolutional Layer: 3**
Filter Size	$S = 8 \sim 256ms$
Filter Strides	$P = \frac{S}{2} = 4$
Number of Filters	128
Activation Function	ReLU
Regularization	L_2

Table 4.4: Description of filter size for the Deep Convolutional Neural Network Architecture in Figure 4.5. The input layer has N samples.

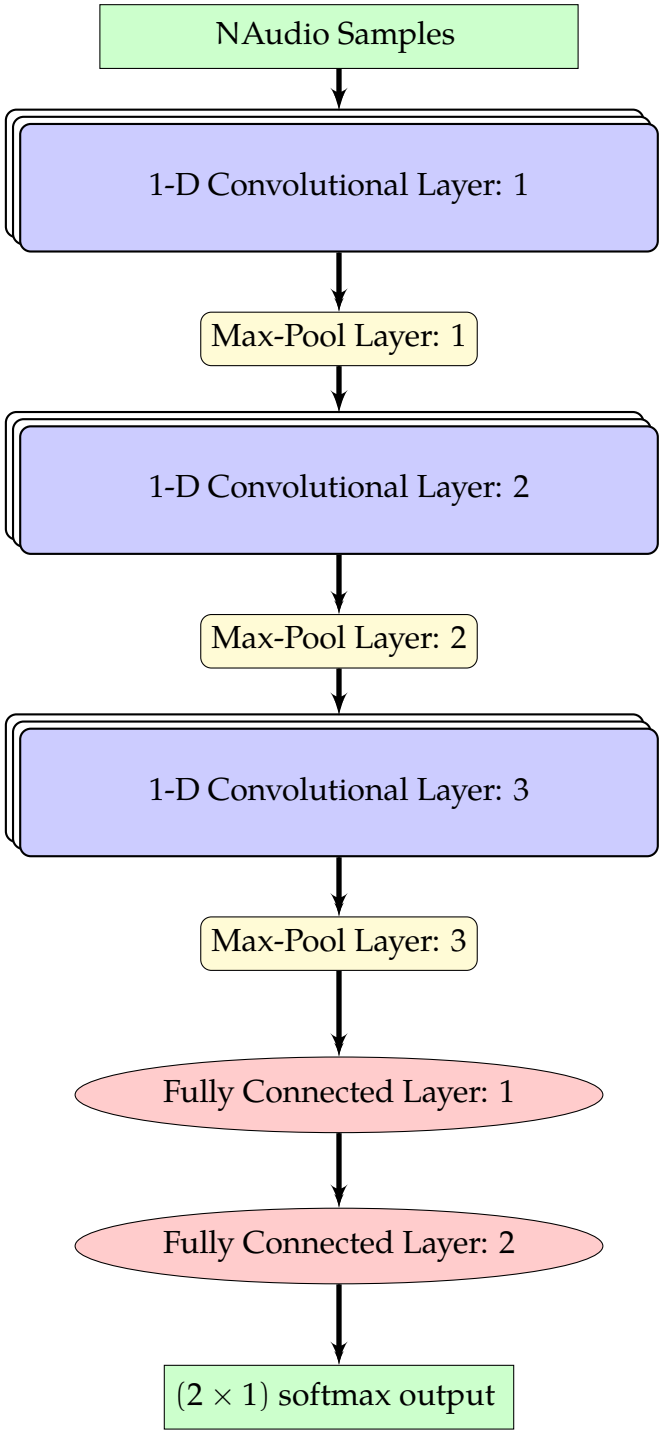


Figure 4.5: Architecture of the Deep Convolutional Neural Network for Keyphrase Recognition

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.01	**0.83**	0.01	0.01	0.0	0.0	0.01	0.0	0.01
hey siri	0.01	0.05	**0.64**	0.01	0.0	0.0	0.09	0.03	0.02
hey cortana	0.01	0.03	0.03	**0.66**	0.04	0.02	0.02	0.02	0.02
nine one one	0.0	0.0	0.0	0.0	**0.83**	0.03	0.0	0.01	0.01
call mom	0.01	0.0	0.01	0.06	0.03	**0.94**	0.0	0.05	0.0
take a picture	0.01	0.02	0.03	0.01	0.0	0.0	**0.77**	0.0	0.01
play a song	0.02	0.01	0.03	0.02	0.01	0.04	0.01	**0.71**	0.03
make a note	0.02	0.03	0.02	0.02	0.01	0.03	0.03	0.01	**0.67**

Table 4.5: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: no obfuscation*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.07	**0.52**	0.06	0.12	0.09	0.1	0.08	0.08	0.1
hey siri	0.1	0.14	**0.53**	0.14	0.12	0.09	0.14	0.13	0.11
hey cortana	0.06	0.09	0.08	**0.58**	0.07	0.08	0.11	0.08	0.11
nine one one	0.02	0.04	0.04	0.03	**0.77**	0.12	0.02	0.08	0.05
call mom	0.01	0.01	0.03	0.04	0.05	**0.83**	0.02	0.1	0.06
take a picture	0.01	0.05	0.03	0.07	0.06	0.03	**0.38**	0.09	0.05
play a song	0.0	0.01	0.01	0.03	0.03	0.05	0.01	**0.39**	0.02
make a note	0.04	0.06	0.08	0.06	0.1	0.08	0.05	0.08	**0.44**

Table 4.6: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: hamming reduction on every 1 sample*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.04	**0.79**	0.06	0.12	0.03	0.03	0.07	0.05	0.05
hey siri	0.06	0.07	**0.45**	0.08	0.04	0.05	0.07	0.04	0.05
hey cortana	0.01	0.07	0.04	**0.57**	0.01	0.02	0.08	0.03	0.05
nine one one	0.02	0.03	0.04	0.03	**0.64**	0.15	0.02	0.07	0.03
call mom	0.0	0.0	0.02	0.02	0.05	**0.71**	0.01	0.05	0.05
take a picture	0.03	0.05	0.02	0.05	0.01	0.01	**0.63**	0.02	0.02
play a song	0.04	0.05	0.07	0.06	0.13	0.1	0.06	**0.32**	0.09
make a note	0.0	0.01	0.01	0.02	0.02	0.03	0.02	0.04	**0.52**

Table 4.7: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: hamming reduction on every 4 samples*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.03	**0.85**	0.0	0.09	0.0	0.0	0.01	0.0	0.0
hey siri	0.0	0.0	**0.99**	0.0	0.0	0.0	0.0	0.0	0.0
hey cortana	0.01	0.02	0.02	**0.91**	0.01	0.01	0.0	0.02	0.03
nine one one	0.01	0.0	0.0	0.01	**0.98**	0.0	0.0	0.01	0.0
call mom	0.0	0.0	0.01	0.0	0.0	**0.81**	0.0	0.01	0.0
take a picture	0.0	0.0	0.0	0.0	0.0	0.0	**0.7**	0.0	0.0
play a song	0.0	0.0	0.0	0.01	0.01	0.01	0.0	**0.68**	0.0
make a note	0.02	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**0.87**

Table 4.8: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: hamming reduction on every 16 samples*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.03	**0.73**	0.03	0.02	0.01	0.0	0.05	0.01	0.02
hey siri	0.01	0.03	**0.51**	0.03	0.01	0.02	0.05	0.02	0.03
hey cortana	0.02	0.05	0.03	**0.74**	0.02	0.02	0.05	0.03	0.01
nine one one	0.04	0.03	0.02	0.07	**0.87**	0.04	0.03	0.03	0.02
call mom	0.0	0.0	0.02	0.02	0.04	**0.92**	0.0	0.08	0.02
take a picture	0.02	0.02	0.04	0.01	0.0	0.0	**0.49**	0.01	0.01
play a song	0.01	0.01	0.1	0.04	0.03	0.13	0.01	**0.81**	0.04
make a note	0.04	0.06	0.03	0.03	0.04	0.03	0.02	0.04	**0.58**

Table 4.9: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: samplerate reduced to 2000 Hz*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.01	**0.34**	0.01	0.03	0.01	0.01	0.02	0.01	0.03
hey siri	0.06	0.06	**0.06**	0.06	0.06	0.06	0.06	0.06	0.06
hey cortana	0.01	0.04	0.07	**0.68**	0.03	0.01	0.03	0.08	0.04
nine one one	0.01	0.01	0.02	0.01	**0.64**	0.02	0.02	0.02	0.01
call mom	0.01	0.02	0.02	0.06	0.04	**0.62**	0.01	0.03	0.02
take a picture	0.05	0.05	0.05	0.05	0.05	0.05	**0.05**	0.05	0.05
play a song	0.04	0.05	0.11	0.03	0.09	0.11	0.03	**0.66**	0.06
make a note	0.02	0.04	0.03	0.08	0.03	0.02	0.06	0.06	**0.26**

Table 4.10: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: samplerate reduced to 1000 Hz*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.03	**0.49**	0.05	0.06	0.03	0.02	0.04	0.02	0.02
hey siri	0.07	0.07	**0.43**	0.1	0.1	0.1	0.09	0.1	0.07
hey cortana	0.02	0.06	0.04	**0.44**	0.01	0.01	0.06	0.01	0.02
nine one one	0.03	0.04	0.05	0.04	**0.72**	0.16	0.02	0.09	0.04
call mom	0.01	0.0	0.02	0.02	0.03	**0.55**	0.02	0.04	0.06
take a picture	0.06	0.05	0.04	0.06	0.01	0.01	**0.63**	0.03	0.02
play a song	0.01	0.0	0.01	0.01	0.05	0.07	0.01	**0.43**	0.05
make a note	0.01	0.03	0.04	0.05	0.04	0.07	0.04	0.04	**0.42**

Table 4.11: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: samplerate reduced to 2000 Hz followed by hamming reduction on every 1 sample*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.05	**0.86**	0.03	0.09	0.03	0.05	0.03	0.04	0.03
hey siri	0.03	0.01	**0.96**	0.03	0.03	0.02	0.04	0.01	0.06
hey cortana	0.01	0.01	0.01	**0.51**	0.01	0.01	0.01	0.01	0.01
nine one one	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0
call mom	0.01	0.0	0.01	0.0	0.0	**0.99**	0.0	0.01	0.0
take a picture	0.01	0.05	0.01	0.03	0.01	0.02	**0.87**	0.01	0.03
play a song	0.02	0.0	0.0	0.0	0.03	0.0	0.01	**0.9**	0.0
make a note	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**1.0**

Table 4.12: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: samplerate reduced to 1000 Hz followed by hamming reduction on every 1 sample*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.01	**0.84**	0.01	0.01	0.02	0.01	0.01	0.02	0.01
hey siri	0.0	0.0	**0.99**	0.0	0.0	0.0	0.0	0.0	0.0
hey cortana	0.04	0.04	0.06	**0.78**	0.01	0.0	0.04	0.02	0.03
nine one one	0.01	0.0	0.0	0.04	**0.91**	0.0	0.01	0.02	0.01
call mom	0.0	0.0	0.0	0.0	0.0	**0.28**	0.0	0.0	0.0
take a picture	0.0	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0
play a song	0.01	0.0	0.0	0.0	0.0	0.0	0.0	**0.98**	0.0
make a note	0.02	0.0	0.01	0.0	0.01	0.03	0.01	0.03	**0.86**

Table 4.13: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: samplerate reduced to 2000 Hz followed by hamming reduction on every 4 samples*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.0	**0.48**	0.0	0.01	0.0	0.0	0.0	0.01	0.0
hey siri	0.03	0.03	**0.72**	0.02	0.03	0.02	0.08	0.01	0.02
hey cortana	0.02	0.06	0.04	**0.77**	0.03	0.03	0.01	0.03	0.11
nine one one	0.02	0.02	0.01	0.01	**0.82**	0.04	0.01	0.01	0.01
call mom	0.01	0.02	0.02	0.01	0.03	**0.1**	0.02	0.03	0.01
take a picture	0.03	0.02	0.01	0.01	0.01	0.03	**0.88**	0.01	0.01
play a song	0.01	0.05	0.03	0.02	0.06	0.05	0.01	**0.92**	0.02
make a note	0.01	0.04	0.04	0.02	0.01	0.01	0.0	0.01	**0.73**

Table 4.14: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: samplerate reduced to 1000 Hz followed by hamming reduction every 4 samples*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.02	**0.72**	0.02	0.06	0.06	0.01	0.06	0.08	0.04
hey siri	0.01	0.05	**0.11**	0.05	0.06	0.1	0.04	0.06	0.08
hey cortana	0.02	0.02	0.01	**0.99**	0.01	0.0	0.01	0.02	0.01
nine one one	0.04	0.05	0.07	0.07	**0.73**	0.12	0.05	0.09	0.08
call mom	0.0	0.01	0.02	0.0	0.02	**0.85**	0.0	0.09	0.04
take a picture	0.0	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0
play a song	0.01	0.0	0.0	0.0	0.0	0.0	0.0	**0.98**	0.0
make a note	0.01	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**1.0**

Table 4.15: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: samplerate reduced to 2000 Hz followed by hamming reduction on every 16 samples*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.0	**1.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0
hey siri	0.0	0.0	**0.76**	0.0	0.0	0.0	0.0	0.0	0.0
hey cortana	0.01	0.0	0.0	**0.99**	0.0	0.0	0.0	0.0	0.01
nine one one	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0
call mom	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0	0.0
take a picture	0.02	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0
play a song	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0
make a note	0.01	0.02	0.0	0.0	0.0	0.05	0.0	0.03	**0.95**

Table 4.16: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: precision reduced to 1-bit*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.03	**0.98**	0.0	0.01	0.0	0.0	0.01	0.0	0.0
hey siri	0.01	0.02	**0.16**	0.03	0.02	0.02	0.03	0.02	0.02
hey cortana	0.0	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0	0.0
nine one one	0.0	0.0	0.0	0.0	**0.9**	0.0	0.0	0.0	0.0
call mom	0.04	0.06	0.06	0.07	0.06	**0.1**	0.07	0.07	0.06
take a picture	0.05	0.04	0.04	0.04	0.05	0.06	**0.49**	0.06	0.05
play a song	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**1.0**	0.0
make a note	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**0.83**

Table 4.17: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: precision reduced to 1-bit followed by hamming reduction on every 4 samples*

None	*random talk*	*okay google*	*hey siri*	*hey cortana*	*nine one one*	*call mom*	*take a picture*	*play a song*	*make a note*
okay google	0.01	**0.79**	0.01	0.01	0.01	0.01	0.0	0.01	0.01
hey siri	0.01	0.04	**0.84**	0.02	0.01	0.02	0.04	0.02	0.01
hey cortana	0.02	0.02	0.03	**0.76**	0.04	0.03	0.03	0.03	0.03
nine one one	0.01	0.06	0.03	0.01	**0.86**	0.03	0.0	0.02	0.06
call mom	0.02	0.01	0.01	0.02	0.02	**0.64**	0.01	0.04	0.02
take a picture	0.06	0.01	0.05	0.03	0.0	0.01	**0.7**	0.02	0.01
play a song	0.08	0.05	0.04	0.08	0.04	0.12	0.05	**0.73**	0.06
make a note	0.02	0.03	0.02	0.02	0.03	0.01	0.01	0.01	**0.68**

Table 4.18: Mean Class Probabilities from 350 audio examples. Each row corresponds to a different keyphrase classification model. Each column corresponds to a different keyphrase audio example. *Obfuscation technique: precision reduced to 1-bit followed by hamming reduction on every 16 samples*

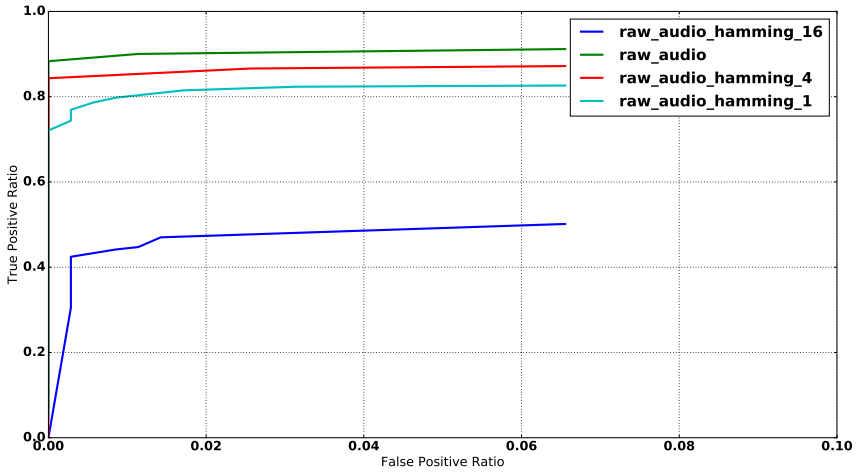


Figure 4.6: Receiver Operating Curve (ROC) for Keyphrase Recognition on Raw Audio and Raw Audio obfuscated with Hamming Reduction

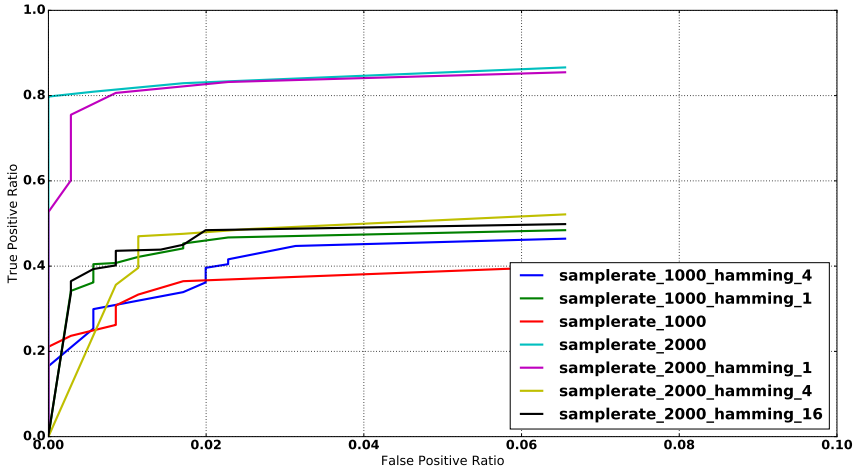


Figure 4.7: Receiver Operating Curve (ROC) for Keyphrase Recognition on Downsampled Audio and Downsampled Audio obfuscated with Hamming Reduction

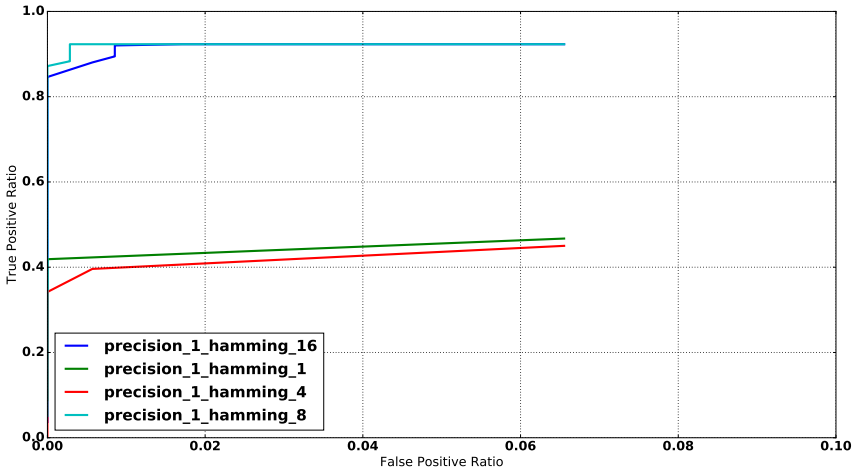


Figure 4.8: Receiver Operating Curve (ROC) for Keyphrase Recognition on Low-precision Audio and Low-precision Audio obfuscated with Hamming Reduction

4.7 Proposed changes to Mobile Keyphrase Recognition

Using the obfuscated audio technique proposed by the results from the dbHound experimental evaluation, I propose that the Android sensor hub applies the obfuscation to microphone audio and broadcasts it to the applications on the device that wish to provide a voice command interface. This model is illustrated in Figure 4.9. This way, applications such as browsers wishing to provide simple voice commands (such as "go back", "open new tab" etc.) will not infringe upon the privacy of the user.

4.8 Conclusion

In this chapter, I present the dbHound system in detail and discuss lightweight obfuscation techniques that can be implemented on mobile devices. Based

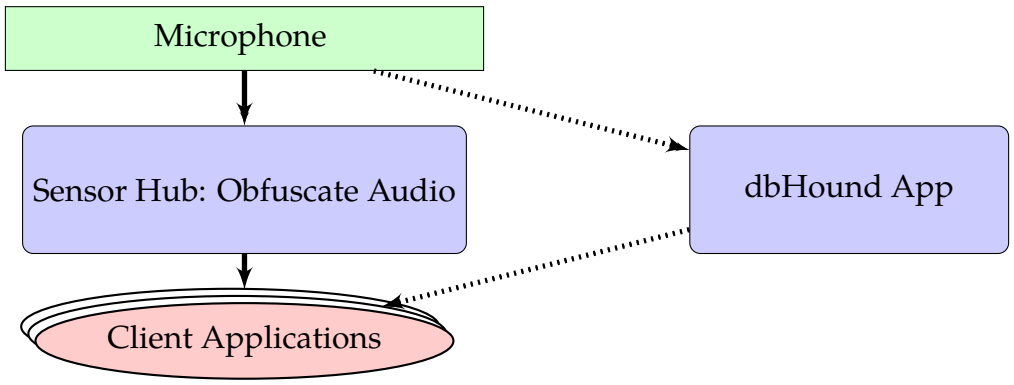


Figure 4.9: Proposed Voice Command Model for Mobile Systems showing dbHound equivalent in parallel

on the experimental data, the most promising obfuscation technique is `precision_1_hamming_16`. In other words, reducing the audio to 1-bit precision and applying the hamming reduce operation over 16 consecutive samples shows the best trade-off between privacy and keyphrase recognition performance.

I envision that such voice-recognition models based on obfuscated audio will provide the solution to maintaining user privacy while improving the accessibility of technology for the physically challenged, as well as providing users with an improved way of interfacing with technology.

5 CONCLUSIONS AND FUTURE DIRECTIONS

5.1 Conclusion

In this thesis, I illustrate applications of ideas from Machine Learning research in three different areas: edge computing systems, wireless systems research and mobile systems research. In all three avenues, there is tremendous scope to define new problems that can be tackled by Machine Learning drive by the volume of data-driven analyses already occurring in practice.

In case of water hardness detection, the machine learning algorithms are already deployed in real systems by virtue of the Madison based startup Emonix Inc [1].

In case of wireless spectrum estimation, this is an actively research unsolved problem with tremendous promise. In my work, I describe the trappings of an initial solution using matrix completion. However, there are several ideas to improve the currently achieved performance, as described in Section 5.2.

In case of privacy preserving keyphrase recognition, to the best of my knowledge, this is the first line of work that examines keyphrase recognition using lightweight audio obfuscation techniques. I hope to carry my initial results here in building a privacy-preserving keyphrase recognition service that can be deployed in mobile systems with minimal effort.

[1] The company can be found online at `https://www.emonix.io`.

5.2 Directions for Future Work

Water Hardness Detection

Characterizing transient water usage patterns

One shortcoming of the existing water flow forecasting approach is that it is unable to predict the instantaneous flow because of the high-order dynamics associated with the signal. Recent research suggests moderately complex LSTMs (Long-Short Term Memory Networks) are able to learn compact representations of high-order dynamics. This may be an interesting avenue to explore in future work.

Collective water quality characterization

With real systems being deployed in improving water softener efficiency, it could be valuable to provide recommendations to the local water authority to improve water quality and resource utility. Using data collected from neighboring installations of the Emonix system, it would be possible to build a database of water quality and usage history with geographic tags. This can be used by the local water authority for understanding drawbacks, if any, in the water distribution infrastructure as well as notify consumers of excessive water usage patterns.

Wireless Spectrum Estimation

Incorporating physical propagation models

One of the drawbacks of the estimation method proposed in this thesis is the 5dBm estimation error. It would be interesting to consider using wireless propagation models along with geographically diverse data (as opposed to using data only from a single location as described in Chapter

3). This would be equivalent to a regularization technique that leverages physical models and allows for more data to factor into the estimation.

Privacy-sensitive Acoustic Perception

Theoretical Analysis based on Differential Privacy

The results from the dbHound system presented in Chapter 4 discuss empirical evaluation of keyphrase recognition from obfuscated audio. However, for strong guarantees on privacy and security, it is important to analyze the obfuscation techniques using the lens of differential privacy. Such an analysis would help to quantify the degree of privacy and security improvements in the context of the threat models described in Section 4.2.

REFERENCES

[1] Alexa and google home record what you say. But what happens to the data? https://www.wired.com/2016/12/alexa-and-google-record-your-voice/. Accessed: 01-15-2017.

[2] Amazon mechanical turk. https://www.mturk.com/mturk/welcome. Accessed: 01-15-2017.

[3] Android sensor hub. https://source.android.com/devices/sensors/sensor-stack.html. Accessed: 01-15-2017.

[4] Cellular Telecommunications and Internet Assocition. http://www.ctia.org/about-us/current-members. Accessed: 2016-10-17.

[5] Explanation of water hardness. https://www.fcwa.org/water/hardness.htm. Accessed: 01-11-2017.

[6] Google Spectrum Database. https://www.google.com/get/spectrumdatabase/. Accessed: 2016-10-18.

[7] Microsoft Whitespaces Database. http://whitespaces.microsoftspectrum.com/. Accessed: 2016-10-18.

[8] Nuts and bolts of applying deep learning. https://kevinzakka.github.io/2016/09/26/applying-deep-learning/. Accessed: 01-15-2017.

[9] Softened water: Benefits Study. http://www.wqa.org/Portals/0/WQRF/ResearchStudy_BenefitsOfSoftenedWater_ExecSummary.pdf. Accessed: 01-11-2017.

[10] Spectrum Bridge Whitespace Database. http://whitespaces.spectrumbridge.com/Main.aspx. Accessed: 2016-10-18.

[11] Cai, Jian-Feng, Emmanuel J. CandÃ¨s, and Zuowei Shen. 2010. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization* 20(4):1956–1982. http://dx.doi.org/10.1137/080738970.

[12] Candès, Emmanuel J., and Benjamin Recht. 2009. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics* 9(6):717.

[13] Chen, Guoguo. 2014. Low resource keyword spotting. *Department of Electrical and Computer Engineering Johns Hopkins University Baltimore, Maryland*.

[14] Chen, Guoguo, Carolina Parada, and Georg Heigold. 2014. Small-footprint keyword spotting using deep neural networks. In *2014 ieee international conference on acoustics, speech and signal processing (icassp)*, 4087–4091. IEEE.

[15] Chen, Yu-hsin, Ignacio Lopez-Moreno, T Sainath, Mirkó Visontai, Raziel Alvarez, and Carolina Parada. 2015. Locallyconnected and convolutional neural networks for small footprint speaker recognition. *Interspeech, Dresden, Germany*.

[16] Dhoutaut, Dominique, Anthony Régis, and François Spies. 2006. Integration of physical phenomena into an experiment-based propagation model. In *Proceedings of the 3rd acm international workshop on performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, 98–105. PE-WASUN '06, New York, NY, USA: ACM.

[17] Foundation, National Science. Today's water softeners. http://standards.nsf.org/apps/group_public/download.php/9248/Softener%20Facts%204-2010%5B1%5D.pdf. Accessed: 01-13-2017.

[18] Green, D., Z. Yun, and M. F. Iskander. 2016. Propagation characteristics in urban environments. In *2016 ieee/aces international conference on wireless information technology and systems (icwits) and applied computational electromagnetics (aces)*, 1–2.

[19] Harrold, T., R. Cepeda, and M. Beach. 2011. Long-term measurements of spectrum occupancy characteristics. In *New frontiers in dynamic spectrum access networks (dyspan), 2011 ieee symposium on*, 83–89.

[20] Klingensmith, Neil, Anantharaghavan Sridhar, Zachary LaVallee, and Suman Banerjee. 2015. Water or slime?: A platform for automating water treatment systems. In *Proceedings of the 2nd acm international conference on embedded systems for energy-efficient built environments*, 75–84. BuildSys '15, New York, NY, USA: ACM.

[21] Kumar, P., N. Rakheja, A. Sarswat, H. Varshney, P. Bhatia, S. R. Goli, V. J. Ribeiro, and M. Sharma. 2013. White space detection and spectrum characterization in urban and rural india. In *World of wireless, mobile and multimedia networks (wowmom), 2013 ieee 14th international symposium and workshops on a*, 1–6.

[22] McHenry, Mark A., Peter A. Tenhula, Dan McCloskey, Dennis A. Roberson, and Cynthia S. Hood. 2006. Chicago spectrum occupancy measurements & analysis and a long-term studies proposal. In *Proceedings of the first international workshop on technology and policy for accessing spectrum*. TAPAS '06, New York, NY, USA: ACM.

[23] Misra, Rakesh, Aditya Gudipati, and Sachin Katti. 2016. Quickc: Practical sub-millisecond transport for small cells. In *Proceedings of the 22nd annual international conference on mobile computing and networking*, 109–121. MobiCom '16, New York, NY, USA: ACM.

[24] Momeni, M., Z. Gharedaghi, M. M. Amin, P. Poursafa, and M. Mansourian. 2014. Does water hardness have preventive effect on cardiovascular disease? *Int J Prev Med* 5(2):159–163.

[25] Olah, Chris. Conv Nets: A modular perspective. `http://colah.github.io/posts/2014-07-Conv-Nets-Modular/`. Accessed: 01-14-2017.

[26] Oppenheim, Alan V, and Ronald W Schafer. 2010. *Discrete-time signal processing*. Pearson Higher Education.

[27] Pan, Xinghao, Maximillian Lam, Stephen Tu, Dimitris Papailliopoulos, Ce Zhang, Michael Jordan, Kannan Ramchandran, Chris Re, and Benjamin Recht. 2016 (to appear). CYCLADES: Conflict-free Asynchronous Machine Learning. In *Advances in neural information processing systems*.

[28] Petrin, Allen, and Paul G Steffes. 2004. Measurement and analysis of urban spectrum usage. In *Proc. of the 2004 international symposium on advanced radio technologies ntia special publication sp-04-409*, 45–48.

[29] Recht, Benjamin, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems 24*, ed. J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, 693–701. Curran Associates, Inc.

[30] Roberson, D. A., C. S. Hood, J. L. LoCicero, and J. T. MacDonald. 2006. Spectral occupancy and interference studies in support of cognitive radio technology deployment. In *Networking technologies for software defined radio networks, 2006. sdr '06.1st ieee workshop on*, 26–35.

[31] Rousseau, Anthony, Paul DelÃ©glise, and Yannick EstÃ¨ve. 2014. Enhancing the ted-lium corpus with selected data for language modeling and more ted talks. In *In proc. lrec*, 26–31.

[32] Sainath, T, and Carolina Parada. 2015. Convolutional neural networks for small-footprint keyword spotting. In *Proc. interspeech*.

[33] Sengupta, P. 2013. Potential health impacts of hard water. *Int J Prev Med* 4(8):866–875.

[34] Taher, T. M., R. B. Bacchus, K. J. Zdunek, and D. A. Roberson. 2011. Long-term spectral occupancy findings in chicago. In *New frontiers in dynamic spectrum access networks (dyspan), 2011 ieee symposium on*, 100–107.

[35] Wikipedia. 2016. Cellular frequencies in the US — Wikipedia, the free encyclopedia. [Online; accessed 17-Oct-2016].

[36] Yin, S., D. Chen, Q. Zhang, M. Liu, and S. Li. 2012. Mining spectrum usage data: A large-scale spectrum measurement study. *IEEE Transactions on Mobile Computing* 11(6):1033–1046.

[37] Zhang, Tan. 2016. Building high-performance wireless systems through dynamic spectrum access. Ph.D. thesis, University of Wisconsin-Madison.

[38] Zhang, Tan, and Suman Banerjee. 2013. Inaccurate spectrum databases?: Public transit to its rescue! In *Proceedings of the twelfth acm workshop on hot topics in networks*, 6:1–6:7. HotNets-XII, New York, NY, USA: ACM.

[39] Zhang, Tan, Ning Leng, and Suman Banerjee. 2014. A vehicle-based measurement framework for enhancing whitespace spectrum databases. In *Proceedings of the 20th annual international conference on*

mobile computing and networking, 17–28. MobiCom '14, New York, NY, USA: ACM.

[40] Zhang, Tan, Ashish Patro, Ning Leng, and Suman Banerjee. 2015. A wireless spectrum analyzer in your pocket. In *Proceedings of the 16th international workshop on mobile computing systems and applications*, 69–74. HotMobile '15, New York, NY, USA: ACM.